SPECIAL C PROGRAMMING ISSUE

April 1988

$3.95 USA (Canada $4.95)

For the PC Systems Integrator

Micro Systems JOURNAL

Hands-On C Programming Tips

- Debugging C Programs
- DOS Functions in C

Turbo C vs. Quick C

Reviews of:

MiniProbe I Debugger
Periscope Debuggers

04

0 38351 19902 9

Powerful *Concurrent*™ DOS 386

Multiuser Multitasking Operating System

MAXIMUM PERFORMANCE . . . Unleash the Power of the 386!

Concurrent™ DOS uses the power of the 386 to efficiently combine its multiuser, multitasking design with the added value of DOS compatibility. Up to ten users can share the resources of a single system through easy-to-connect serial terminals as configured. Digital Research provides tools for those VARs and OEMs needing to expand beyond ten users.

MAXIMUM COMPATIBILITY . . . Runs Multiuser Applications While Simultaneously Running Many Popular PC-DOS Applications.

The applications running on your current system (Lotus® 1-2-3®, dBase® III, WordPerfect® and many more) are still usable and don't have to be replaced with "work-alikes" and "compatibles." Multiple DOS applications can be run from serial terminals while the system console can execute as many as four applications concurrently.

MINIMUM INVESTMENT . . . Protect Your Development Investment with Easy Migration within the Intel® Microprocessor Family.

Take advantage of developed and proven application software that provides solutions for businesses that range from medical practices to manufacturing floors. A library of Concurrent DOS multiuser applications already exists to meet the diverse requirements of many end-user environments.

FEATURES:

- PC DOS 3.3 Compatible.
- Supports IBM® Personal System/2™ Model 80, Compaq® Deskpro® 386 and 100% Compatibles.
- Executes Multiple Applications on Serial Terminals.
- Simultaneously Executes Up to 255 Different Tasks.
- Supports Multiusers Sharing System Resources.
- Easy to Install and Operate.
- Full Complement of Development Tools and Over 1000 Business Solutions Available.
- AT Performance at the Serial Terminals.
- Serial Port Configurability Up to 38.4K Baud.

Call Today and Receive a FREE Poster of Single-User and Multiuser Concurrent DOS Applications.

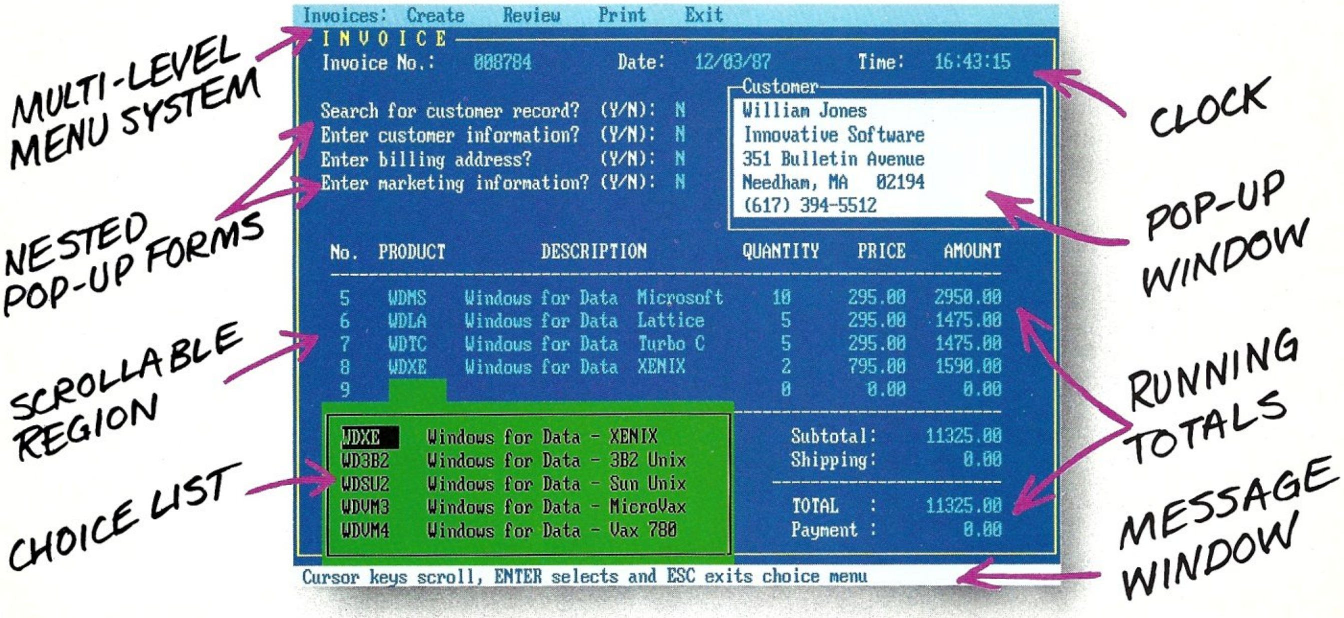
1-800-443-4200.

DIGITAL RESEARCH®

New Prices

OS/2

WINDOWS FOR DATA®

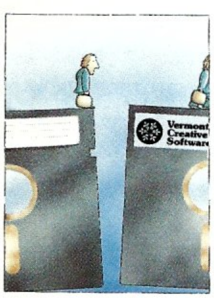


If you program in C, take a few moments to learn how Windows for Data can help you build a state-of-the-art user interface.

☑ **C**reate and manage menus, data-entry forms, context-sensitive help, and text displays — all within windows.

☑ **D**evelop window-based OS/2 programs right now, without the headaches of learning OS/2 screen management. Run the same source code in PCDOS and OS/2 protected mode.

☑ **B**uild a better front end for any DBMS that has a C-language interface (most popular ones do).



FROM END TO BEGINNING

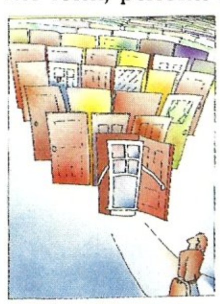
Windows for Data begins where other screen packages end, with special features like nested pop-up forms and menus, field entry from lists of choices, scrollable regions for the entry of variable numbers of line items, and an exclusive built-in debugging system.

NO WALLS

If you've been frustrated by the limitations of other screen utilities, don't be discouraged. You won't run into walls with Windows for Data. Our customers repeatedly tell us how they've used our system in ways we never imagined — but which we anticipated by designing Windows for Data for unprecedented adapatability. You will be amazed at what you can do with Windows for Data.

YOU ARE ALWAYS IN CHARGE

Control functions that you write and attach to fields and/or keys can read, compare, validate, and change the data values in all fields of the form. Upon entry or exit from any field, control functions can call up subsidiary forms and menus, change the active field, exit or abort the form, perform almost any task you can imagine.



OUR WINDOWS WILL OPEN DOORS

Our windows will open doors to new markets for your software. High-performance, source-code-compatible versions of Windows for Data are now available for **PCDOS, OS/2, XENIX, UNIX,** and **VMS.** PCDOS versions are fully compatible with Microsoft Windows. **No royalties.**

MONEY BACK GUARANTEE

You owe it to yourself and your programs to try Windows for Data. If not satisfied, you can return it for a full refund.

Prices: PCDOS $295, Source $295. OS/2 $495. XENIX $795. UNIX, VMS, please call.

Call: (802) 848-7731 **ext. 33**
Telex: 510-601-4160 VCSOFT FAX 802-848-3502

Vermont Creative Software

21 Elm Ave.
Richford,
VT 05476

We've spent years developing our file manager so you won't have to.

You could invest hundreds of programming hours writing a file management system for your next application.

Or you could simply invest in Btrieve.® And get all the file handling functionality you need, through subroutine calls from your favorite programming language.

Portable. Write your application once. Wherever Btrieve runs, your application will run, whether in a single user or multiuser environment. In fact, Btrieve is the standard access to NetWare.®

Fast. Written in assembly language, Btrieve uses b-tree indexing algorithms with caching and automatic balancing for fast, efficient file management.

Safe. Btrieve is the only fault tolerant file manager with built-in file recovery. In the event of a system or power failure, your database is protected.

Flexible. Develop applications with the capabilities you need most. Like 255 open files, unlimited records per file, 24 indexes per file, and a maximum file size of up to four gigabytes. You can access Btrieve from BASIC, C, Pascal, COBOL and others.

Invest in Btrieve. At just $245 for single user and $595 for multiuser, it's a small price to pay for all the file manager you'll ever need.* And you'll never pay royalties on the applications you develop. To find out more, see your authorized Novell reseller, or call (512) 346-8380.

For more information, call from your modem 1-800-444-4472 (8 bit, no parity, 1 stop bit) and enter the access code NVBT13.

NOVELL

For software solutions, you should be seeing red.

*Suggested retail price (US dollars) World Headquarters, 122 East 1700 South, Provo, Utah 84601 (801) 379-5900
Requires PC-DOS or MS-DOS 2.X, 3.X or Xenix.

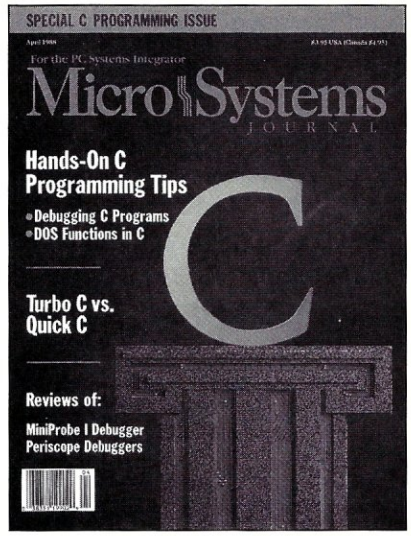
For the PC Systems Integrator
Micro/Systems JOURNAL

April 1988
Vol. 4 . No. 4

FEATURE ARTICLES

PRODUCT REVIEWS

COLUMNS



About the cover: There is an axiom that all PC professionals hold as true: hardware is useless without software. That is why this issue focuses on C software development tools. As processing power continues to grow, there is a greater challenge to adapt and develop programs that will effectively harness that power. On this month's cover, we have placed a C, representing the patriarch of programming languages, on a pedestal of microcircuitry, thus symbolizing the symbiotic relationship between hardware and software.
Cover photograph by Michael Carr

DEPARTMENTS

Micro/Systems Journal (ISSN 8750-9482) is published monthly by M & T Publishing, Inc., 501 Galveston Drive, Redwood City, CA 94063; (415) 366-3600. Second-class postage paid at Redwood City and at additional entry points. **POSTMASTER:** send address changes (form 3579) to Micro/Systems Journal, Box 3713, Escondido, CA 92025. **Change of Address:** Please send old label and new address to Micro/Systems Journal, Box 3713, Escondido, CA 92025. **Customer Service:** For subscription problems, call: (800) 321-3333; In California call (619) 485-9623. **Subscription Rates:** U.S., $29.95 for one year, $56.97 for two years. Canada and Mexico add $17 per year for airmail, $7 per year for surface mail. Other countries add $28 per year for airmail.

From the Editor's Desk

Software Battles—Ready For Prime Time TV?

PC users are being treated to battles in the software marketplace that are beginning to rival the battles we see on the evening TV soaps. Who knows, they might become the basis for a new TV series that will top the likes of "Dallas," "Falcon Crest," and "Dynasty." After all, those shows are based on the competition in businesses like oil, wine, and construction, but computer people can't relate to those goings on.

What we hackers need is a series we can relate to! I think it is time for CBS, NBC, or ABC to start a soap series based on characters modeled after the likes of Bill Gates and Phillipe Kahn, cloners such as Adam Osborne, and upstart shareware marketers such as Jim Button and Bob Wallace. And there are the hardware battles, too. Just think of the battles royal between IBM, Apple, DEC, AT&T, and the like, and the competing operating systems. Takeovers, bankruptcies, sellouts, alliances—it's all there, waiting for the scriptwriters.

The material for plots is incredible. We could have courtroom scenes in which lawyers argue and witnesses testify about the "look and feel" of user screen interfaces and the copying of microcode. We could have scenes showing what goes on behind the closed hotel room doors at big industry trade shows; board meetings where the Chairman of the Board is ousted and replaced by the company President he himself hired; and the scene could quickly switch to a messy garage where two teenagers, dressed in torn jeans and sandals, are making product breakthroughs that elude multinational companies. And then there are things like illegal program copying, vaporware, bug fixing, compatibility, etc. We have ready-made plots for a series that will lure us away from our system screen to our TV screen and keep us on the edge of seat. We could have five or six plots running concurrently with enough material to go on for years.

Actually, I already have the script for the first episode of a series on disk and I am trying to peddle it to several TV networks. I will let you know how I make out.

Happy April.

Sol Libes

For the PC Systems Integrator
Micro/Systems Journal

EDITORIAL

Founder and Editor Sol Libes
Technical Editors Stephen R. Davis, Don Libes
Associate Editors Lennie Libes, Susan Libes
Contributing Editors A.G.W. Cameron, Michael Cherry, P.L. Olympia
Consulting Editor Michael Swaine
Managing Editor Thomas M. Woolf

PRODUCTION

Art & Production Director Larry Clay
Art Director Kobi Morgan
Assistant Art Director Barbara Mautz
Typesetter Lorraine Buckland

CIRCULATION

Director of Circulation Maureen Kaminski
Subscription Supv. Kathleen Shay
Newsstand Sales Coordinator Sarah Frisbie
Fulfillment Coordinator Francesca Martin

ADMINISTRATION

Vice President Finance & Operations Kate Wheat
Business Manager Betty Trickett
Accounting Supv. Mayda Lopez-Quintana
Accounts Payable Asst. Luanne Rocklewitz
Accts. Receivable Asst. Wendy Ho

ADVERTISING

Advertising Director Richard Mixter
National Account Mgr. Dwight Schwab
National Account Mgr. Tami Brenton (415) 886-1957
Advertising Coordinator Shaun Hooper

M&T Publishing, Inc.

Chairman of the Board Otmar Weber
Director C.F. Von Quadt
President and Publisher Laird Foshay
V.P. of Publishing William P. Howard

Micro/Systems Journal (ISSN 8750-9482) is published monthly by M & T Publishing, Inc., 501 Galveston Drive, Redwood City, CA 94063; (415) 366-3600. Second-class postage paid at Redwood City and at additional entry points.

Article Submission: If you have a specific area of expertise or interest and would like to contribute, please write Micro/Systems Journal, P.O. Box 1192, Mountainside, NJ 07092; (201) 522-9347, or contact M & T Publishing, Inc., 501 Galveston Drive, Redwood City, CA 94063; (415) 366-3600. Please do not submit articles without first contacting the editors. Author's guidelines available upon request.

Correspondence: Please send letters to the editor to Micro/Systems Journal, 501 Galveston Drive, Redwood City, CA 94063. Other editorial correspondence may also be directed to P.O. Box 1192, Mountainside, NJ 07092. The editors may also be reached via MCI Mail (SLIBES or MSJ).

Advertising Rates: Available upon request. Call (415) 366-3600 or write to: Advertising Department, Micro/Systems Journal, 501 Galveston Drive, Redwood City, CA 94063.

News & Views

by Sol Libes

Random Gossip & Rumors

IBM is rumored to be ready to discontinue its PC Convertible Model III and introduce new 286 and 386 portables. After three previous failures, IBM will again attempt to compete with **Compaq** in the portable marketplace. And there are rumors that IBM is planning to have another go at the consumer computer market with a "PS/2jr"—will they get it right this time? Another company with this number of duds would have gone out of business.

AT&T has purchased a 20 percent interest in **Sun Microsystems.** This will make AT&T Sun's largest stockholder and give Sun a terrific and much-needed financial shot in the arm. Sun and AT&T had previously entered into an agreement whereby Sun committed to support AT&T's UNIX efforts and AT&T committed to the use of Sun's new Sparc CPU.

Intel delivered a paper at the February International Solid State Circuits Conference describing a new RISC-type microprocessor chip set. Although still in development, it includes many system-level functions previously handled by operating system software.

Apple is rumored to be far along in the development of a laptop Mac, with an introduction possible as early as the second quarter of this year. And **Tandy Corporation** is rumored working on a Macintosh clone.

Microsoft is expected to shortly release a version of its C compiler that will include a full-screen editor and OS/2 tools among other enhancements. And a new version of the Bascom Basic compiler also is expected that includes these same features.

Microsoft reports that OS/2 Version 1.1 containing the Presentation Manager is now in beta test and is expected to be shipping copies to OEMs in October. IBM has announced that it will ship OS/2 Version 1.1 to dealers in October as well.

And look for Microsoft to release Version 1.0 of the LAN Manager in July. It is a completely new product, which beta testers report is vastly improved over the old DOS-based MS-Net system. Both DOS and OS/2 systems can coexist on the net. It will be an OEM product (*à la* OS/2) and sold by OEMs, such as 3Com.

The PS/2 Compatibles Are Coming

Look for several vendors to introduce clones of PS/2 Models 50, 60, and 80 that are compatible with IBM's patented Micro Channel bus architecture at next month's Comdex show in Atlanta. These units are expected to have faster clock speeds (look for Model 80 clones with 25-MHz clocks), larger and higher performance disk drives, more memory tightly coupled to the CPU, memory caching, improved VGA, and other features that are not available on the IBM units. Clone vendors are rumored to include **Kaypro**, **Compaq**, and smaller companies using chips and board designs from **Chips & Technology** and **Western Digital.** These companies are known to be working on obtaining PS/2 licensing agreements from IBM. However, it is not yet clear what steps IBM will take to protect its products. Production shipments of these units are expected as early as June. IBM anticipating introduction of these units recently instituted price cuts on its units. IBM also is expected to respond shortly with upgraded PS/2 systems.

C&T has already shown PS/2 prototype boards for the Models 50 and 60 that have almost half as many chips (from 119 down to 68) as the IBM boards. And C&T sources claim that its Model 80 (80386) board has only 66 ICs compared to 179 on IBM's board.

OS/2—Early Reports

Early reports from OS/2 users indicate that OS/2 has some problems that will require additional work on the part of IBM and Microsoft. Multitasking is reported to be several times slower than comparable operations run under Xenix on the same hardware. Also, OS/2 cannot reliably handle 9,600-baud communications via serial ports when multitasking.

Users who have attempted to bring up IBM's implementation of OS/2 on non-IBM systems report that, in many cases, it will not even boot, or when it does boot there may be problems with software that works with peripheral devices. OS/2 bypasses the AT system's BIOS ROM making OS/2 a more stringent test of hardware compatibility. Device driver software must be provided for system boards and plug-in board products.

Compaq has reported that IBM's OS/2 runs without modification on all of its 286/386-based systems. Nonetheless, Compaq plans to introduce its own version. Users who want to run OS/2 on AT clones and find that the vendor does not plan to release a version of OS/2 for their systems may find that they will have to try implementations from different vendors in the hope of finding one that will run on their system. If none can be found, they will find themselves shut out from running OS/2 on their AT clones.

Microsoft has indicated that it is already at work on OS/2 improvements. In the works are support for multiple processors and improved memory management for 386-based systems. The PS/2 has a multiprocessing bus architecture, but as yet no plug-in products take advantage of it and neither DOS nor the current version of OS/2 provide for processor arbitration.

Floppy Software Density Increases

Many PC application software packages have grown so large in size that some suppliers are now furnishing them in archived (compressed) form on floppy disks along with software to unarchive the programs and do automatic installation. The result is a typical savings of 20–40 percent in disk cost as some packages would otherwise require as many as 20 floppy disks.

Vendors of software for OS/2 are furnishing their software on high-density 1.4-MB disks. When this software is archived, more than 2MB of software can be shipped on one floppy disk.

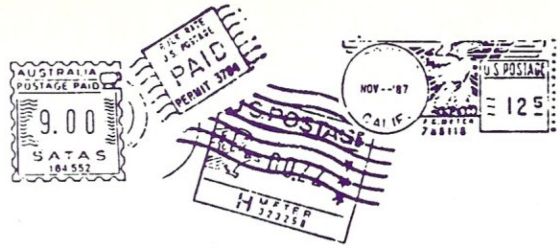
386 Systems In High Demand

Dataquest, a market researcher that tracks PC sales, predicts that roughly one million 386 systems will be sold this year. The 386 share of the marketplace will also equal that of 8088-based systems. 286-based systems now occupy the largest share of the PC market. The likelihood is that 386-based systems will overtake 8088-based systems next year.

286-based systems are now considered the system for general use and are expected to be the dominant system this year and next. 386-based systems are chosen when speed is important—for power hungry applications such as CAD, desktop publishing, network file servers, and multiuser system hosts.

IBM and **Compaq** are the dominant 386 system suppliers, with Compaq reported to have a slight edge over IBM by virtue of its early introduction of a 386-based system. Also, Compaq has stayed ahead by introducing a 20-MHz machine with added features. □

there is mail . . .



In Search of the Missing Code

To the editor,

I observed with horror that the listings that accompanied my article in the January 1988 issue ("A Hardware Breakout Switch for PC-DOS's Debug," page 46) looked as though it had been through a demented text formatter. The result probably stemmed from an incompatibility between WordStar's tab handling in non-document mode and *Micro/System's* layout word processor. Nonetheless, I'm sure that most readers should be able to decipher the listing.

However, coupled with the fact that the last eight lines were left off the published listing, I suspect even Champollion himself would have trouble working it out. I hope the attached partial listing (Listing 1) will help those who wish to type the listing into their computer.

Alex Cameron
Surrey Downs, Australia

From the Editor: Our apologies to our readers and to Mr. Cameron. The problem is, indeed, one of translation that has since been resolved. For those who are interested, the corrected source code is available from M&T Publishing on an MS-DOS disk, as well as on CompuServe (type GO DDJ FORUM).

That Was Then, This Is Now

Dear Micro/Systems*:*

Your February issue arrived in my mailbox this morning and I thought again of the first time I saw *Micro/ Systems Journal.*

That was quite a while ago (maybe three years?), early enough in our company's history that we were still dazzled at being able to afford a real office. That *Micro/Systems Journal* was a very slim, cheaply printed family effort that wanted to give Ziff-Davis a run for its money. It was interesting, well-written, and certainly spunky, but I didn't have great hope for its survival.

I congratulate you. Your *Journal* has survived and thrived. I know it wasn't easy.

Elizabeth G. Bryson
President
Golden Bow Systems
San Diego, California

PC-Plus Has Been Improved

Dear Micro/Systems*:*

We appreciated the article by Dr. A.L. Bender on Alloy's PC-Plus network (*Micro/Systems,* January 1988). We would like to add a few comments to update the information in the article.

In addition to the QICSTOR-Plus and 12-slot X-bus, Alloy now provides a 4-slot expansion bus. The 4- and 12-slot expansion buses are optionally available with a new version of the HI card—the HI/2—that permits the expansion buses to be attached to the Micro Channel bus of IBM's new Personal System/ 2, Models 50, 60, and 80. Alloy's PC-Plus also operates directly with the hard disks delivered with IBM's PC/XTs and ATs, and most compatibles—it is not necessary to use Alloy add-on hard disks as Dr. Bender did.

We have introduced a new Slave/286 card, featuring an 8-MHz 80286 processor and one megabyte of RAM for users requiring additional computational power. The Slave/286 can be operated in systems with the V20-based PC-Slave/16N, allowing CPU requirements to be matched to user needs.

Alloy has made arrangements with the Harris Corporation for national field service. A variety of services are available, including hardware installation, extended warranty plans, on-site service, or carry-in depot repair. Users should contact Harris Corporation directly at (214) 620-4440.

David Friesen
Director of Strategic Marketing
Alloy Computer Products, Inc.

Listing 1. The End of Alex Cameron's Hardware Breakout Switch Program

```
            MOV     DV,OFFSET END_OF_RESIDENT_CODE+1
            INT     27h
;
NO_INSTALL:
;
            INT     20H
BANNER      DB      'BREAKNMI - NMI,Keyboard and Timer Vectors'
            DB      ' installed.',10,13,'$'
;
BREAKNMI    ENDP
CSEG        ENDS
            END         BREAKNMI
```

A Short-cut for Network Programming with Clipper

Dear Micro/Systems*:*

I read with interest the recent article "Using dBASE III+ and Clipper A86 With Novell Netware 286," by Henry J. Franzoni III (*Micro/Systems,* January 1988).

That Mr. Franzoni should write such an article with no reference to NetLib, the most popular network add-in for Clipper, is amazing; like omitting UI programmer from an article about dBASE program generators.

The features of NetLib are far too numerous to describe here. It includes all of the features mentioned in Mr. Franzoni's article, plus many station numbers, user ID, semaphore locks, print spooler control, deadlock avoidance, journaling (with the latest release), and many more. All features are accessed by an easy use of syntax and require only a few lines of coding by the programmer. For example, Mr. Franzoni's article outlines a rather laborious procedure for swapping network printers. Under NetLib, the same task is accomplished by one function call:

```
N_SPOOL("=2")
   && switch to 2nd printer
```

In addition, NetLib supports many networks and multiuser systems that Clipper alone will not support, such as the Alloy PC-Slave (also featured in *Micro/Systems,* January 1988).

While I can certainly be accused of blowing my own horn, I feel that it is important that your readers know the scope of the tools available to them. Keep up the good work.

Neil Weicher
President
Communication Horizons
New York, N.Y.

From the Editor:

We work very hard to provide the most complete information we can for our readers, including the latest tools as well as the latest techniques. Thanks for the update. We are always interested in hearing more about products that improve productivity, and will profile as many of them as space and resources permit.

4 TIMES FASTER THAN TODAY'S FASTEST ASSEMBLER!

That's right. 4 times faster.

Clocking in at over 75,000 lines per minute on a 6MHz IBM AT, OPTASM is four times faster than Microsoft's MASM 5.0. **4 times faster — that's 400% more throughput!**

But speed is only one part of it. OPTASM is nearly 100% compatible with MASM 5.0 (except 386 support).
It is the only single assembler capable of supporting the various incompatibilities between MASM 3, 4 & 5. That makes OPTASM **more MASM compatible than any single version of MASM!**

Other features? OPTASM generates smaller code without ever generating extra NOP's. It automatically handles jumps out of range, up to 15,000 symbols and most of MASM's phase errors. It also boasts a built in MAKE and simplifies segmentation.

That's why we can make our OPTASM challenge: Test OPTASM head to head against MICROSOFT MASM 5.0. Order **both** assemblers with their 30-day guarantees. In a lot less than 30 days, you'll see just how dazzling OPTASM's speed really is. You'll realize that we're compatible, easier to use, and deliver many more important features than MASM. So accept our challenge. Try both assemblers. **Four times faster and more features, too.** We know which one you'll send back.

Write or call us to order or for our detailed brochure.

OPTASM: $195 Guaranteed returnable within 30 days.

WHAT DO PROGRAMMERS SAY ABOUT OPTASM?

"It (OPTASM) just blows MASM away . . . reduces my assemble time for Periscope from 3-plus minutes to less than 45 seconds."
Brett Salter, President,
The Periscope Company

"OPTASM has been absolutely solid. For me, the most useful new product in 1987."
Chris Dunford, Columbia, MD

SLR SYSTEMS

1622 N. Main Street
Butler, PA 16001
412-282-0864
BBS 412-282-2799
Telex 559215
800-833-3061

by Don Libes

The International Obfuscated C Code Contest

It's April, and that could only mean one thing—time to present the 1987 International Obfuscated C Code Contest winners!

If you've never heard of the IOCCC, you are in for a real treat. The International Obfuscated C Code Contest is run annually by Landon Noll (Amdahl Corp) and Larry Bassel (National Semiconductor) who collect C code that is so awful to read, it is actually funny. Viewed in the right light, you might even call it educational. The 1987 winners are presented toward the end of the column. The results of the first, second, and third contests were published in *Micro/Systems Journal* in September/October 1985, May/June 1986, and March/April 1987, respectively, if you want to track them down. (They are worth it!)

The 1988 contest is now open. Here are the rules:

Goals of the Contest:

- To write the most Obscure/Obfuscated C program under the rules below.
- To show what should NOT be done in C programs.
- To provide a safe forum for poor C code.

Rules:

To help us handle the vast volume of entries, we ask that you follow the rules below. (Sorry for the length, but we need all the help we can get!)

1. Your source must be 1536 bytes or less, and it must be a complete program, not just a subroutine.
2. To help us process your entries, we ask that you submit entries in the following format. (Please be sure to include the - - - lines, otherwise our extraction program may skip your entry!):

```
---header items---
name: Your name, of course!
org: School/Company/
Organization
e-mail address: e-mail address
from a well known site
postal address: Postal address,
include your country as well
environment: Indicate Hardware
& OS under which program was
tested
remarks: (Only item that is
required: see below)
---how to compile---
XGive command(s) needed to
Xcompile your program.
XFollow same rules as
Xgiven for program below
Xexcept that command size
Xmust be 160 characters or
Xless.
---program---
XPlace obfuscated source of
X1536 characters or less in
Xthis section. Add a leading
X"X' to each line to avoid
Xproblems with mailers.
XSome mailers don't like
Xfiles with very long lines.
XIf your entry contains lines
longer than 80 characters
we ask you to form con
tinuation line sets. To
form a coCntinuation line
set, place a 'C' character
at the point of a split
and place a 'C'C (instead
of an X) at the beginning
of the next line. Finally,
end the continuation line
set as normal.
XThe C\nC's and leading X's
Xwill be removed prior to
Xextraction and thus they
Xdon't contribute toward
Xthe source character count.
XAll other characters are
Xconsidered to be source.
XNew lines count as 1
Xcharacter. Assume a stand-
Xard 8 character tab stop.
---end---
```

3. Regarding the header items:
   - Any text outside of the above format will be kept confidential. (The form of the header items is not strict.)
   - The "remarks" item is *not* optional. Please include:

     what this program does;
     why you think the program is obfuscated;
     any indicate remarks you wish to make.
4. Your entry should be written in common C (K&R and common extensions).
5. The program must be an original work. All programs must be in the public domain; copyrighted programs will be rejected.
6. Entries must be received between 25-Mar-88 0:00 GMT and 25-May-88 0:00 GMT. E-mail entries to:

   `...!amdahl!obfuscate`

   Amdahl talks to hplabs, decwrl, pyramid, seismo and cbosgd. We will attempt to e-mail a confirmation of receipt of contest entries, however, since e-mail is not reliable, you may not receive it. Although, people are encouraged to submit entries via e-mail, one may mail entries to the following postal address:

   Landon Curt Noll
   Amdahl Corp.
   1250 E. Arques Ave., M/S 316
   P.O. Box 3470
   Sunnyvale, CA 94088-3470
   U.S.A.

   Write the words: "International Obfuscated C Code Contest" near the bottom left corner of the envelope.
7. Each person may submit up to five entries. Multiple entries must be sent in separate e-mail letters or postal envelopes.

Announcement of Winners:

- The first announcement will be at the Summer '88 Usenix BOF.
- An announcement will be posted to *mod.announce* near mid-June 1988 stating to which newsgroup the winners have been posted.
- An article containing the winning entries will be published in a future issue of *Micro/Systems.*
- Winners receive international fame and flames!

Judging:

Awards will be given to the best entry in a number of categories. The actual category list will vary, depending on the types of entries we received. As a guide, consider using the following categories:

- The best, small, one-line program
- The most obscure algorithm
- The strangest source layout
- The most useful obfuscated program
- The most creatively obfuscated program

- Anything else so strange that it deserves an award

Points to Ponder:

People are encouraged to examine winners of the previous contests. A copy of these entries was posted to *mod.sources* on or about March 12, 1987.

Contact the *mod.sources* moderator if you missed that article. Keep in mind that rules change from year to year, so some winning entries may not be valid entries this year. What was unique and novel one year might be "old" the next year, so use your judgment.

We examine each entry on several levels of confusion. For example, each entry is judged when we:

- look at the original source
- run it through:

```
sed -e ',^#[  ]*
define,d' | /lib/cpp
```

- run it through a C beautifier
- examine the algorithm
- compile and lint it
- execute it

One-line programs are best when they are short, obscure, and concise.

We tend to dislike programs that:

- are very hardware specific
- are very OS or UNIX version specific (index/strchr differences are okay, but socket/streams specific code is likely not to be)
- dump core or have compiler warnings (it is okay only if you warn us in the "remark" header item)
- won't compile under both BSD or SYSV UNIX
- use an excessively long compile line to get around the size limit
- simply carries an idea to excess without reason
- are similar to previous winners
- are similar to previous losers

Simply abusing *#defines* or *-Dfoo=bar* won't go as far as a more well-rounded program.

We like programs that:

- do something quasi-interesting
- pass *lint* without complaint
- are portable
- are unique or novel in their obfuscation style
- are concise
- use size to do something interesting or that use size to introduce several different types of obfuscation
- make us laugh or throw up.

Some types of programs can't excel in some areas. We try to account for this by giving awards to programs in a number of areas. Of course, your program doesn't have to excel in all areas, but doing well in a few helps.

Listing 1. In the category of "Best Obfuscator of Programs," the winner is Paul Heckbert of Pixar.

```
#include <ctype.h>
#include <stdio.h>
#define _ define
#_ A putchar
#_ B return
#_ C index
char*r,c[300001],*d=">=<=!===||&&->++-->><<",*i,*l,*j,*m,*k,*n,*h,*y;e,u=1,v,w,
f=1,p,s,x;main(a,b)char**b;{p=a>1?atoi(b[1]):79;r=c+read(0,j=l=i=c,300000);v=g(
j,&m);for(k=m;v!=2;j=k,m=n,v=w,k=m){w=g(k,&n);if(v==1&&m-j==1&&*j==35)e&&A(10),
e=f=0;if(!f&&v==3&&(char*)C(j,10)<m)A(10),e=0,f=1;else if(v>2&&(u||w)&&(f||u)&&
(l-i>1||*i!=61||n-k>1||!C("-*&",*k)))continue;else if(v==3)if(f&&e+1+n-k>p&&e)A
(10),e=0;else A(32),e++;else{if(f&&e+m-j>p&&e)A(10),e=0;e+=m-j;k=j;while(k<m)A(
*k++);}i=j;l=m;u=v;}e&&A(10);}g(j,m)char*j,**m;{if(j>=r)B*m=j,2;s=isdigit(*j)||
*j==46&&isdigit(j[1]);for(h=j;h<r;h++)if(!isalnum(*h)&&*h!=95&&(!s||*h!=46)&&(!
s||h[-1]!=101&&h[-1]!=69||!C("+-",*h)))break;if(h>j)B*m=h,0;x=1;for(h=j;h<r&&C(
" \t\n",*h);h++);if(h>j)h--,x=3;if(*j==34||*j==39)for(h=j+1;h<r&&*h!=*j;h++)if(
*h==92)h++;for(y=d;*y&&strncmp(y,j,2);y+=2);if(*y)h=j+1;if(!strncmp("/*",j,2)){
h=j+2;while(*++h!=42||*++h!=47);x=4;}*m=h+1;B x;}
```

Listing 2. In the category of "Most Useful Obfuscation," the winner is Larry Wall of Unisys, System Development Group, Santa Monica, Calif.

```
#define iv 4
#define v ;(void
#define XI(xi)int xi[iv*'V'];
#define L(c,l,i)c(){d(l);m(i);}
#include <stdio.h>
int*cc,c,i,ix='\t',exit(),X='\n'*'\d';XI(VI)XI(xi)extern(*vi[])(),(*
signal())();char*V,cm,D['x'],M='\n',I,*gets();L(MV,V,(c+='d',ix))m(x){v)
signal(X/'I',vi[x]);}d(x)char*x;{v)write(i,x,i);}L(MC,V,M+I)xv(){c>=i?m(
c/M/M+M):(d(&M),m(cm));}L(mi,V+cm,M)L(md,V,M)MM(){c=c*M%X;V-=cm;m(ix);}
LXX(){gets(D)||(vi[iv])();c=atoi(D);while(c>=X){c-=X;d("m");}V="ivxlcdm"
+iv;m(ix);}LV(){c-=c;while((i=cc[*D=getchar()])>-I)i?(c?(c<i&&l(-c-c,
"%d"),l(i,"+%d")):l(i,"(%d")):(c&&l(M,")"),l(*D,"%c")),c=i;c&&l(X,")"),l
(-i,"%c");m(iv-!(i&I));}L(ml,V,'\f')li(){m(cm+!isatty(i=I));}ii(){m(c=cm
=++I)v)pipe(VI);cc=xi+cm++;for(V="jWYmDEnX";*V;V++)xi[*V^' ']=c,xi[*V++]
=c,c*=M,xi[*V^' ']=xi[*V]=c>>I;cc[-I]-=ix v)close(*VI);cc[M]-=M;}main(){
(*vi)();for(;v)write(VI[I],V,M));}l(xl,lx)char*lx;{v)printf(lx,xl)v)
fflush(stdout);}L(xx,V+I,(c-=X/cm,ix))int(*vi[])()={ii,li,LXX,LV,exit,l,
d,l,d,xv,MM,md,MC,ml,MV,xx,xx,xx,xx,MV,mi};
```

Listing 3. In the category of "Best Layout," the winner is Brian Westley of Starfire Consulting, St. Paul, Minn.

```
                         char rahc
                            [ ]
                             =
                          "\n/"
                             ,
                         redivider
                            [ ]
                             =
                "Able was I ere I saw elbA"
                             ,
                             *
                      deliver,reviled
                             =
                            1+1
                             ,
                        niam ; main
                            ( )
                           {/*\}
                            \*/
                          int tni
                             =
                            0x0
                             ,
                      rahctup,putchar
                            ( )
                   ,LACEDx0 = 0xDECAL,
                         rof ; for
                      (;(int) (tni);)
                        (int) (tni)
                  = reviled ; deliver =
                         redivider
                             ;
for ((int)(tni)++,++reviled;reviled* *deliver;deliver++,++(int)(tni)) rof
                             =
                      (int) -1- (tni)
                  ;reviled--;--deliver;
                      (tni)  =  (int)
                  - 0xDECAL + LACEDx0 -
                         rof ; for
     (reviled--,(int)--(tni);(int) (tni);(int)--(tni),--deliver)
                     rahctup = putchar
                    (reviled* *deliver)
                             ;
                     rahctup * putchar
                     ((char) * (rahc))
                             ;
                            /*\
                           {\*/}
```

The judging will be done by Landon Noll and Larry Bassel. If you have any questions or comments, please feel free to send them to: . . . !amdahl!-judges

The 1987 winners

First, try to understand the program by just reading the source and the judges comments. Then, try running the program. If you are still confused, try sending the source through the C Preprocessor, or a good C beautifier (unlike the BSD indent(1) program, which dumped core processing some of the entries). Should you give up, next month's column will present explanations of all the programs.

Assume entries did not pass *lint* unless stated otherwise.

Note that several entries had lines so long that they had to be broken up in order to fit in the magazine.

The envelope please...

In the category of "Best Obfuscator of Programs," the winner is Paul Heckbert of Pixar (Listing 1).

Judges comments: On SYSV systems, compile with: *-Dindex=strchr.* To compile on a 16-bit machine, change 300000's to 30000. Passes *BSD lint.* Try:

```
ph 40 < ph.c > foo.c; cc foo.c -o
    ph
ph 20 < a_C_prog.c > bar.c; cc
    bar.c
```

Read and compile *foo.c.* We used this program to help us examine contest entries that caused BSD's fold(1) program to choke. Thank you, Paul, I have added your program to our obfuscated C contest tool set.

In the category of "Most Useful Obfuscation," the winner is Larry Wall of Unisys, System Development Group, Santa Monica, California (Listing 2).

Judges comments: Join all the lines together except for the first five. Passes *BSD lint.* Try:

```
lwall | bc | lwall
input:  x*x
input:  c^2
```

Also try:

```
lwall | bc   and   lwall | cat
```

For a good time, try to understand why Larry calls the signal routine in this program. Larry gives some credit to his brother-in-law, Mark Biggar, for this crazy use of signals.

In the category of "Best Layout," the winner is Brian Westley of Starfire Consulting, St. Paul, Minn. (Listing 3).

Judges comments: *Putchar* must exist in the C library and not just as a macro. If it fails to compile, add the line: *#include <stdio.h>* at the top of the program. Passed *BSD lint* (probably due to a bug in *BSD lint*).

Line-by-line symmetry performed better than any C beautifier. Think of if it as a C Inkblot.

In the category of the "Best One Liner," the winner is David Korn of AT&T Bell Labs, Murray Hill, New Jersey (Listing 4).

Judges comments: Passes *BSD lint.*

Compile on a UNIX system, or at least using a C implementation that fakes it. This program may not be valid under the proposed ANSI C standard. See if your can understand what is does before you run the program.

Landon interviewed someone who claimed to be a hot C programmer. After reviewing this little program, the person cooled his reputation a bit.

David Korn's */bin/ksh* provides us with a greatly improved version of the */bin/sh.* The source for V7's */bin/sh* greatly inspired this contest.

In the category of "Best Abuse of the Rules," the winner is Mark Biggar, Unisys, System Development Group, Santa Monica, California:

```
P;
```

Judges notes: compile with:

```
cc -DC="R>0" -DI="if(T)O"
-DO="c=write(1,&c,1);"
   -DP="main(){X}"\
   -DR="read(0,&c,1)"
-DT="c!=015" -DW="while(C)I"
   -DX="char c;W" markb.c
```

Passes *BSD lint.* At least one version of lint is thrown into an infinite loop by this entry. Try:

```
... | markb | od -c
   (remember to compile as
    indicated above)
```

By changing the compile line, you can make this program do anything you want. This is a very efficient way to transfer source code, although it increases the size of Makefiles.

With only slight variations, this program can be set to many uses. Consider how easy it would be to release UNIX source in this form. So what if the make files increase a bit!

One vendor's *lint* got hung in an infinite loop over this entry!

In the category of "Worst Style," the winner is Spencer Hines of OnLine

Listing 4. In the category of the "Best One Liner," the winner is David Korn of AT&T Bell Labs, Murry Hill, N.J.

```
main() { printf(&unix["\021%six\012\0"],(unix)["have"]+"fun"-0x60);}
```

Listing 5. In the category of "Worst Style," the winner is Spencer Hines of OnLine Computer Systems, Hyattsville, Md.

```
#include <stdio.h>
#include <malloc.h>
main(togo,toog)
int togo;
char *toog[];
{char *ogto,   tgoo[80];FILE  *ogot;  int    oogt=0, ootg,  otog=79,
ottg=1;if (    togo==  ottg)   goto   gogo;  goto    goog;  ggot:
if (   fgets(  tgoo,   otog,   ogot)) goto   gtgo;   goto   gott;
gtot:  exit(); ogtg: ++oogt;   goto   ogoo;  togg:   if (   ootg > 0)
goto   oggt;   goto    ggot;   ogog:  if (  !ogot)   goto   gogo;
goto   ggto;   gtto:   printf( "%d    goto   \'s\n", oogt); goto
gtot;  oggt:   if (   !memcmp( ogto, "goto", 4))     goto   otgg;
goto   gooo;   gogo:   exit(   ottg); tggo:  ootg=   strlen(tgoo);
goto   tgog;   oogo: --ootg;   goto   togg;  gooo: ++ogto;  goto
oogo;  gott:   fclose( ogot);  goto   gtto;  otgg:   ogto=  ogto +3;
goto   ogtg;   tgog:   ootg-=4;goto   togg;  gtgo:   ogto=  tgoo;
goto   tggo;   ogoo:   ootg-=3;goto   gooo;  goog:   ogot=  fopen(
toog[  ottg],  "r");   goto   ogog;  ggto:  ogto=   tgoo;  goto
ggot;}
```

Listing 6. And for the grand prize of "Most Well Rounded Confusion," the winner is Roemer B. Lievaart of VU Infomatica, Amsterdam, The Netherlands.

```
#define D define
#D Y return
#D R for
#D e while
#D I printf
#D l int
#D C y=v+111;H(x,v)*y++= *x
#D H(a,b)R(a=b+11;a<b+89;a++)
#D s(a)t=scanf("%d",&a)
l V[1100],u,r[]={-1,-11,-10,-9,1,11,10,9},h[]={11,18,81,88},ih[]={22,27,72,77},
bz,lv=60,*x,*y,m,t;S(d,v,f,a,b)l*v;{l c=0,*n=v+100,bw=d<u-1?a:-9000,w,z,i,zb,q=
3-f;if(d>u){R(w=i=0;i<4;i++)w+=(m=v[h[i]])==f?300:m==q?-300:(t=v[ih[i]])==f?-50
:t==q?50:0;return w;}H(z,0){if(GZ(v,z,f,100)){c++;w= -S(d+1,n,q,-b,-bw);if(w>bw
){zb=z;bw=w;if(w>=b||w>=8003)Y w;}}}if(!c){bz=0;C;Y-S(d+1,n,q,-b,-bw);}bz=zb;Y
d>=u-1?bw+(c<<3):bw;}main(){R(;t<1100;t+=100)R(m=0;m<100;m++)V[t+m]=m<11||m>88
||(m+1)%10<2?3:0;V[44]=V[55]=1;V[45]=V[54]=2;I("Level:");s(u);e(lv>0){do{I("Yo\
u:");s(m);}e(!GZ(V,m,2,0)&&m!=99);if(m!=99)lv--;if(lv<15&&u<10)u+=2;I("Wait\n")
;I("Value:%d\n",S(0,V,1,-9000,9000));I("move: %d\n",(lv-=GZ(V,bz,1,0),bz));}}GZ
(v,z,f,o)l*v;{l*j,q=3-f,g=0,i,h,*k=v+z;if(*k==0)R(i=7;i>=0;i--){j=k+(h=r[i]);e(
*j==q)j+=h;if(*j==f&&j-h!=k){if(!g){g=1;C;}e(j!=k)*((j-=h)+o)=f;}}Y g;}
```

Computer Systems, Hyattsville, Maryland (Listing 5).

Judges comments: Try:

```
avalon avalon.c
```

Specifically designed to maximize the bother function for structured programmers, this program takes *goto* statements to their logical conclusion. The layout and choice of names are classic.

We consider this to be a beautiful counter-example of Frank Rubin's letter to ACM form titled: " "GOTO Considered Harmful" Considered Harmful" (see *Communications of the ACM,* March 1987, Page 195–196).

The grand prize, "Most Well-Rounded Confusion," is awarded to Roemer B. Lievaart of VU Informatica, Amsterdam, The Netherlands (Listing 6).

Sample session (user input surrounded by angle brackets "<" and ">"):

```
Level:<2> (value from 0 to 10)
You:<34>  (value from 11 to 88)
Wait
Value:-16
move:33
You:<43>
Wait
Value:-32
move:35
You:<36>
Wait
Value:16
move:64
You:<65>
Wait
Value:0
move:37
...
```

Continue as needed. If you get stuck, try 99.

This program is confusing in what it looks like, what it does, and how it does it. This is yet another winner from The Netherlands! □

Don Libes is a computer scientist working on artificial intelligence in robot control systems in the Washington, D.C. area.

THE BEST OF BOTH WORLDS

Developing an application used to be easy — all you had to do was program it. But today, with countless languages, compilers, libraries, databases, editors, debuggers, and other tools, it is choosing the right development software that creates the real problem.

Now *The Andsor Collection* introduces a unique solution: a collection of sophisticated development tools, which you can use on their own, or together with your old ones.

The Andsor Collection is, of course, the superb application development system that programmers, VARs, and other developers have been using for over two years. And starting with Version 2.2, *The Andsor Collection* has acquired a new dimension: now you can access all its functions from within another program!

Think of it as a comprehensive, universal, language independent library. But *The Andsor Collection* is not a collection of subroutines: it is a seamless, integrated, interactive environment, specifically designed to expedite application development.

Whether you use C, Pascal, Cobol, Fortran, Basic, or any other language, *The Andsor Collection* can enhance your applications dramatically. Whether you add functions to an old program, or you write a new one, you can make them faster, more efficient, and more appealing.

Use *The Andsor Collection* to implement an entire application, or just portions of an application. You can, for example, create a windowed environment, add attractive data entry functions, define indexed data file structures, produce sophisticated reports or forms, and so on.

System Features

The Andsor Collection is the most versatile application development environment. And when using it with your programs, all its countless features can become part of your application. Hundreds of commands, functions, and options, are available to help you implement any application.

No list can be complete, so here are just some of these features: powerful database functions, maintenance-free multi-index data files, variable length data fields, unlimited file relations, complete window management, unique text processing functions, flexible data tables, powerful inquiry and reporting functions, versatile data entry capabilities, flexible procedural language, automatic error handling, extensive computational capabilities, data analysis and statistics, unique programmable charts, many printing functions, data communications, convenient system log file, full control over color attributes. And much, much more.

How It Works

This is simple and ingenious. Your program and *The Andsor Collection* reside in memory together (you load *The Andsor Collection*, which then loads your program). To transfer control to those portions of the application implemented in *The Andsor Collection*, you simply issue a software interrupt in your program, exactly the way DOS and BIOS functions are called. (If you are not familiar with this, examples in the manual show you how to do it.)

While in *The Andsor Collection*, the operation is identical to its operation when used alone. Finally, one command returns control to your program. And if you need this, simple commands transfer data directly to and from memory areas in your program (a number of formats are possible, all in standard ASCII, compatible with any language). Both *The Andsor Collection* and your program run as ordinary DOS programs. And you can also use them with other software (such as permanently resident programs).

Although *The Andsor Collection* has far more features than other development systems, it is only one tenth their size. So the entire system can stay in memory, keeping all functions instantly accessible.

And *The Andsor Collection* is famous for its unique interactive environment. There is no conversion or translation — modify a procedure, a file definition or relation, a data entry screen, or anything else, and the change takes effect immediately, even while the application is running! Application development is a new experience.

The application users will benefit too. *The Andsor Collection* is amazingly fast, and since all data is in variable length format, the files take a fraction of the space needed with other systems. So not only will you develop your applications sooner, but they will be more efficient too. Whether you use *The Andsor Collection* alone, or to enhance a program.

So get the best of both worlds. Order *The Andsor Collection* today, and discover a whole new environment, without giving up your old development tools or your existing applications. Moreover, *The Andsor Collection* will be useful with all your future applications and languages.

You won't find a better value in development software: one program that is both a powerful stand-alone application development system, and a unique language independent collection of software tools; plus the run-time interpreter with unlimited royalty-free distribution. All for an incredibly low price. And with our 60 day money back guarantee*, you have little to lose and a lot to gain.

"We looked at several systems and decided to standardize on The Andsor Collection. We are using it in many applications in several departments."

Gordon Eyers, Director of Information Services, Public Utilities Commission, Brantford, Ontario

"With The Andsor Collection we have achieved faster development and more efficient applications, which is important in large and complex projects like our Court Management System."

Dr. Mark Schrager, Consultant, Municipal Computer Services, Rochester, New York

"The Andsor Collection is unequalled when we need a solution in a hurry. Applications that we have implemented include modeling, data acquisition and analysis, and reporting."

Joe Blask, Engineering Support, General Motors, Troy, Michigan

ANDSOR®

The Andsor Collection™

ANDSOR RESEARCH INC.
390 Bay Street, Suite 2000
Toronto, Ontario M5H 2Y2
(416) 245-8073

To order call toll free
(U.S. and Canada)

1-800-628-2828
Ext. 535

$145 *(includes shipping*)*

Visa, MC, AmEx, Check

*Price includes shipping in the U.S. and Canada. Please add $10 for shipping to other countries. If you return the software, $8 will be deducted from the refund, to cover our shipping cost.

System requirements: any IBM PC or PS/2 or fully compatible, 250K+ (excluding DOS and other programs), one disk drive or hard disk, monochrome or color monitor, DOS 2.0+ or OS/2

by Stephen R. Davis

Virtual Memory Techniques: Part 2— Pointer Functions

In the initial discussion of Virtual Memory Managers (see *Micro/ Systems*, March 1988), I compared arrays of arrays with arrays of pointers to arrays. I demonstrated how they are similar in use, and how arrays of pointers to arrays are flexible. This time, I will investigate the topic further. But first I want to clarify a few things and discuss some new Turbo Pascal products.

Exaggeration

My November/December 1987 column, "VAR Variables," was devoted to the passing of arrays to procedures, both as VAR and as non-VAR variables. *Micro/Systems Journal* readers were not asleep, and a few of them challenged the assertions I made in that column.

Some readers questioned my claim that "the Turbo manual implies that array variables are always passed by reference." These readers pointed out that the Version 3.0 manual emphatically states this on page 224. Yes, but it is nonetheless incorrect, at least for Version 3.01A of the Turbo compiler. A detailed examination of the assembly language code generated by Turbo reveals that arrays not declared as VAR parameters are pushed in their entirety onto the called program's stack by a library routine called explicitly for that purpose.

Another reader, J. C. Blanford of Dothan, Alabama, reacted to my concluding remark: "if performance means anything . . . use VAR variables." Blanford's timing program contained two procedures, Change and VChange, both of which accessed a large array, one as a "normal" and one as a VAR parameter. The main timing loop was designed to call either one or the other of these two routines. By editing the call and rerunning the program, the user could quickly compare the timing of the two otherwise identical procedures.

When I executed Blanford's example program passing an array of 1,000 integers, I found, to my chagrin, virtually identical timings for Change and VChange! I repeated my earlier tests and they showed that VAR parameters were significantly faster. After some reflection (and a peek at the assembly language code involved), I realized that my earlier claim had been slightly exaggerated. The problem is that passing an array as a "normal" parameter means the entire array must be copied onto the stack. Although Turbo does this using a comparatively fast block-move machine instruction, this is still time consuming. Having copied the entire array, however, subsequent accesses are somewhat faster than VAR parameter accesses, since the segment register is not reloaded with each access and the instructions are slightly faster. Both Change and VChange access each element of the array once. Consequently, the penalty paid for copying the array to the stack is made up by the faster access time.

When I modified Change and VChange so that only 10 of the 1,000 elements of the array *A[ ]* are accessed, the results were different. Here, VChange is approximately 100 times faster than Change. This mimics the tests that I had performed, which accessed representative elements from the array. Making the array larger only heightens the effect. When the size of *A[ ]* is increased to 10,000, the ratio is closer to 1,000 times.

However, if the called procedure accesses each element of *A[ ]* more than once, the penalty paid in copying the array onto the stack is compensated for by the slower accesses. When Change and VChange are modified to access each element of *A[ ]* some 100 times, the "normal" Change procedure actually finishes some 20 percent faster than VChange.

Therefore, if each element of an array is to be accessed by a procedure more than once, it may actually prove faster to pass the array normally rather than as a VAR variable. There is a third possibility: refer to the array globally rather than passing it as a parameter. Although this method is aesthetically less pleasing, it suffers neither from the copy penalty nor from the slower accesses and proves slightly faster than either of the other methods.

Turbo Products

Several new Turbo Pascal products have appeared since my last column (*MS/J*, March 1988). The Turbo Power people are at it again, this time introducing a set of utilities to make bridging the gap between Turbo 3.0 and 4.0 easier. One of these is an Overlay Manager that restores to 4.0 an overlay capability similar to that found in 3.0. Anyone porting overlayed 3.0 code to the new Turbo standard should

Listing 1. Returning the address of a record for Turbo Pascal 4.0

```
{ Simplistic Function Pdata does nothing more than return
address of Index'th record of an array of records. Main
program uses Pdata first to initialize array and then
demonstrates that program is working as designed.}

Type
     data = record
               contents: array [0..9] of integer
            end;
     ptr  = ^data;

Var
     mdata : array [0..9] of data;
     i,j   : integer;

Function Pdata (index : integer) : ptr;
begin
     Pdata := ptr(@mdata [index])
end;

begin
     { Initialize array of records
       to known values }
     for i := 0 to 9 do
          for j := 0 to 9 do
               pdata(i)^.contents[j] := i * j;

     { Now print out a few representative values
       both conventionally and using Pdata for
       comparison }
     for i := 0 to 9 do
          writeln ('i = ', i,
                  '    mdata[i].contents[i] = ',
                       mdata[i].contents[i]:3,
                  '    pdata(i)^.contents[i] = ',
                       pdata(i)^.contents[i]:3)
end.
```

give them a call.

Turbo Power has also been showing off a late beta version of TDebug designed for Turbo 4.0. This really dynamite debugger is an extension of the company's TDebug-Plus for 3.0. Anyone comfortable with that debugger will feel right at home with this 4.0 version. The windows are snappy and attractive and the interface is natural and powerful. Added to this version is a Codeview-like assembly language mode that intermixes Pascal and the resulting assembly language statements—very educational. Also added are improved video support and keyboard macros for commands that are often repeated. Bugs just don't have a chance with this package.

I also received is a very nice shareware debugging aid from Paradigm systems, TPCV. This program allows TP 4.0 programs to be debugged with the Microsoft Codeview debugger.

Codeview is Microsoft's flagship debugger, which is distributed with their MS-DOS compilers and assembler. Probably more for marketing than technical reasons, Codeview accepts its symbol information directly from the .EXE file rather than from the load map. Turbo Pascal does not include this information in the .EXE file it generates. TPCV takes the public symbols from the load map generated by Turbo Pascal's TPMAP and places them into the .EXE file in Codeview format!

To use TPCV, simply compile with the detail load map option turned on, generate a load map with TPMAP, run TPCV, and then execute Codeview on the program. All of the Codeview commands are available for use. Local symbols are not available, since these do not appear in the load map, but all public variables and procedure names, and most line numbers are available. Surprisingly, Codeview did not appear to have any problem with Turbo Pascal units. There may be a few small glitches, however, since Codeview was not designed for Turbo Pascal

Today's Topic—Pointer Functions

In quest of a virtual memory manager (VMM) for Turbo Pascal programs, last time we examined how it is often simpler and faster to handle a large matrix as an array of pointers to arrays rather than simply an array of arrays. The syntax resembled:

```
Type
    example = array [1..10]
    of integer;
    exptr = ^example;
Var
    mdata : array [1..10] of
    example; {array of arrays}
    pdata : array [1..10] of
    exptr;  {array of ^arrays}
```

Individual integers are referenced as:

```
     mdata[i][j]  := ...

     pdata[i]^[j] := ...
```

It is just as simple to envision an array of pointers to records, the syntax of the declaration appearing:

```
Type
    example = record
      idata : array [1..10] of
      integer;
      cdata : array [1..10] of
      character
            end;
    exptr = ^example;

Var
    mdata : array [1..10] of
    example;
    pdata : array [1..10] of
    exptr;
```

with references appearing as:

```
mdata[i].idata[j] := ...
 pdata[i]^.idata[j] := ...
```

The Plot Thickens

If *pdata* were a function that returned the address of an array of arrays or records, its use under Turbo 4.0 would be similar to that above:

```
pdata(i)^[j]        := ...
 for an array}
 pdata(i)^.idata[j] := ...
 for a structure}
```

Notice the round braces following *pdata* since this is a function call. Also notice that the function call appears on the left side of the equal sign. This causes no problems because it is not the value returned by the function that is the target at the equal sign, but rather the memory location to which that value points.

The program shown in Listing 1 is a simple example of such a function. The variable *mdata* is an array of records of type *data*. The function *pdata* accepts a simple integer index and returns a pointer to the "index'th" record in *mdata* (thus, *pdata(i)^* becomes functionally equivalent to *mdata[i]*). The first part of the program uses *pdata* on the left hand side of the equal sign to initialize the array of records to some known value. The second part prints out some representative values. By comparing *mdata[i]* to *pdata(i)^*, we are reasonably certain that our program is working as expected.

This gives us the mechanism we need to implement our VMM in Turbo 4.0. We break up our huge data structure into manageably sized records, each referenced by an index like an array. The user-program accesses these blocks by invoking the VMM function, which returns the location of that block in memory. If the block is not currently in memory, the VMM can load it from disk and return it to the address where it was loaded. The user program treats this like a reference to a simple matrix or array of structures.

Note, however, that 3.0 only accepts a variable name in front of the indirection carat. Therefore, to execute the program listed under 3.0, replace

```
pdata(i)^.contents[j]
```

with

```
temp := pdata(i);
temp^.contents[j];
```

where temp has been declared to be of type *ptr*. Also replace the expression *ptr(@mdata[index])* with *addr-(mdata[index])*.

Conclusion

Now we have the tool we need to implement our Virtual Memory Manager—the function which returns a pointer to either a record or a simple array. In my next column, I will discuss how such a function keeps track of these records. □

Stephen Randy Davis is technical editor for Micro/Systems, *and a programmer for a defense contractor in Greenville, Texas.*

Product Information

Turbo Power Software
3109 Scotts Valley Dr., Ste. 122
Scotts Valley, CA 95066
(408) 438-8608

Paradigm Systems
P. O. Box 152
Milford, MA 01757
(617) 543-3609

Assistance in Debugging C Programs

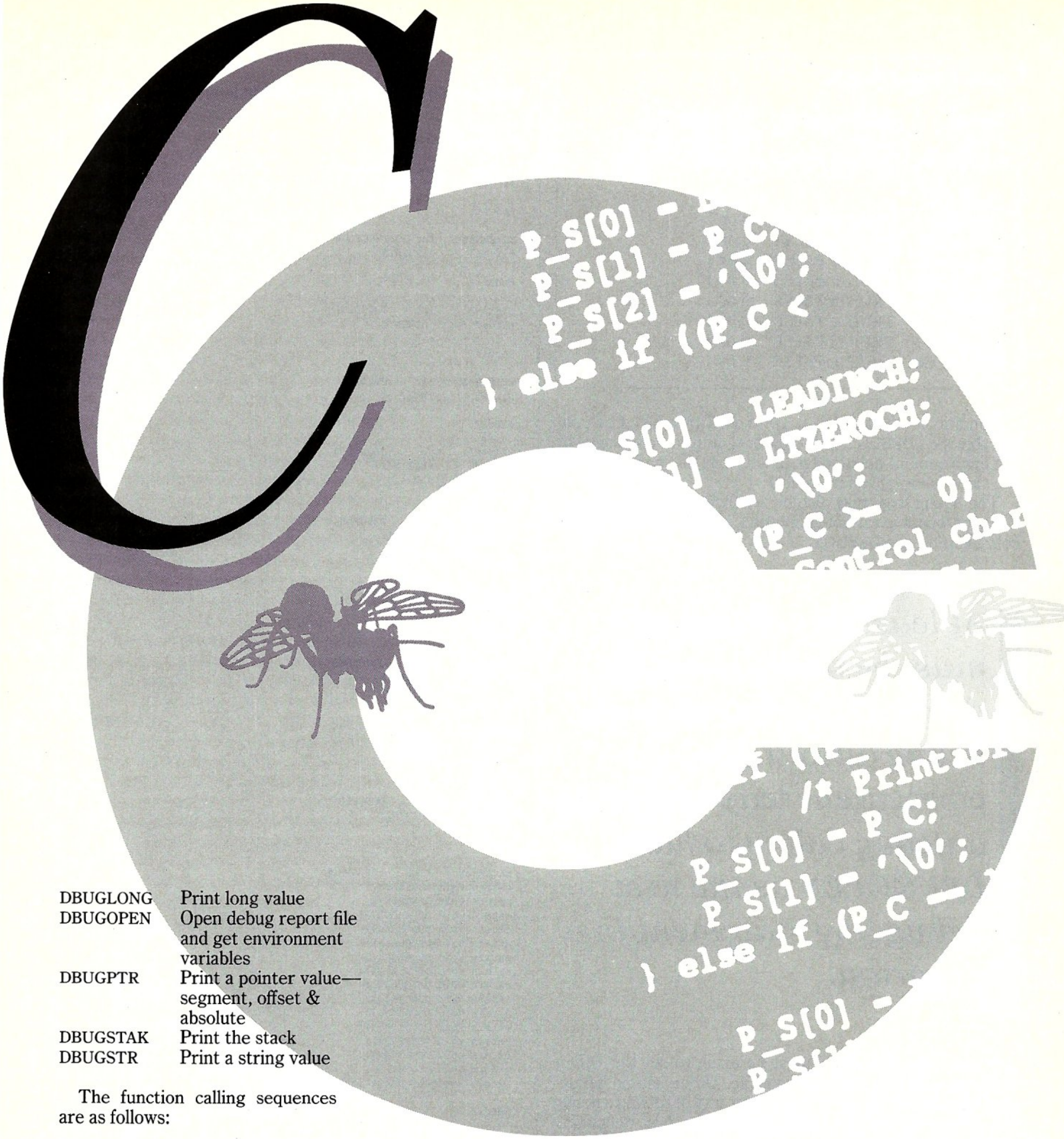
by Bill Rogers

Spot program bugs using printf *statements and issue a debugging report to a printer or file.*

When debugging, a common technique used by C programmers is to add *printf* statements to a function being debugged. These *printf* statements assist in analyzing flow and in spotting variables assuming incorrect values. Since this practice is so common, a formal approach to adding these debugging statements and to printing or displaying the results can reduce the programming time and also the debugging time. The approach illustrated here offers a debugging tool as outlined in the header DEBUG.H (Listing 1), the program DEBUG.C (Listing 2), and the support program BASEPTR.ASM (Listing 3). When properly used, these functions cover flow analysis, function input parameters display, function output parameters display, and function return value display. Intermediate results may also be displayed. (The debugging report may be directed to a file rather than the screen so that the report does not interfere with normal screen input and output.)

The debugging package contains the following functions:

Function	Description
DBUGBEGN	Function entry
DBUGBIN	Print int in binary
DBUGBOOL	Print boolean value (i.e., int that is 0 or 1)
DBUGCHAR	Print character value
DBUGDBL	Print double value
DBUGEND	Function exit
DBUGFLOT	Print float value
DBUGHEX	Print int value in hex
DBUGINT	Print int value



DBUGLONG	Print long value
DBUGOPEN	Open debug report file and get environment variables
DBUGPTR	Print a pointer value—segment, offset & absolute
DBUGSTAK	Print the stack
DBUGSTR	Print a string value

The function calling sequences are as follows:

```
DBUGOPEN();

DBUGBEGN(FUNC,SCCSID);
    FUNC     function name, string
    SCCSID   file name & version, string

DBUGEND(FUNC);
    FUNC     function name, string

DBUGSTAK(FUNC,NUM);
    FUNC     function name, string
    NUM      number of stack words, integer

DBUGxxxx(FUNC,VARNAME,VARVAL);
    FUNC     function name, string
    VARNAME  variable name, string
    VARVAL   variable value, appropriate
                 variable type matching xxxx
```

Table 1. Debug Flags.

Debug Flag	Environment Value	C External Variable Value
DBUGFLG0	TRUE or FALSE	0 or 1 (if 1 output to "stderr")
DBUGFLG1	TRUE or FALSE	0 or 1
DBUGFLG2	TRUE or FALSE	0 or 1
DBUGFLG3	TRUE or FALSE	0 or 1
DBUGFLG4	TRUE or FALSE	0 or 1
DBUGFLG5	TRUE or FALSE	0 or 1
DBUGFLG6	TRUE or FALSE	0 or 1
DBUGFLG7	TRUE or FALSE	0 or 1
DBUGFLG8	TRUE or FALSE	0 or 1

Conditional compilation is controlled by a preprocessor variable (i.e., DEBUG_1). Conditional output is controlled by the environment debug flags (and C external variables with the same name) given in Table 1.

The debug flags may also be set from a file by replacing the preprocessor variable ENVFLAG with FILEFLAG and recompiling the file DEBUG.C.

At least one debug call must be made to set the debug flags and open the debug output file ... A convenient way of setting a debug call is to have DBUGOPEN as the first debugging statement in a program.

At least one debug call must be made to set the debug flags and open the debug output file. Otherwise no debugging output will appear. A convenient way of setting a debug call is to have the call DBUGOPEN as the first debugging statement in a program.

The approach to adding debugging statements by means of a predefined scheme carries the following benefits:

1. Adding debugging statements takes less time;
2. Fewer errors are made by introducing the debugging statements;
3. The output report is in a uniform format;
4. The mechanism for conditional compilation and conditional debugging output is well defined;
5. Displaying pointers, which can become a little involved, are already defined;
6. Displaying stack snapshots, which can be more than an little involved, are already defined; and
7. A production version of a program can be produced without deleting any code.

Listing 1. DEBUG.H file; debug definitions

```
#ifndef _debug
#define _debug

EXTERNAL    int     DBUGOPFL
#ifdef      INTRINIT
            = 0
#endif
            ;
EXTERNAL    int     DBUGFLG0
#ifdef      INTRINIT
            = 0
#endif
            ;
EXTERNAL    int     DBUGFLG1
#ifdef      INTRINIT
            = 0
#endif
            ;
EXTERNAL    int     DBUGFLG2
#ifdef      INTRINIT
            = 0
#endif
            ;
EXTERNAL    int     DBUGFLG3
#ifdef      INTRINIT
            = 0
#endif
            ;
EXTERNAL    int     DBUGFLG4
#ifdef      INTRINIT
            = 0
#endif
            ;
EXTERNAL    int     DBUGFLG5
#ifdef      INTRINIT
            = 0
#endif
            ;
EXTERNAL    int     DBUGFLG6
#ifdef      INTRINIT
            = 0
#endif
            ;
EXTERNAL    int     DBUGFLG7
#ifdef      INTRINIT
            = 0
#endif
            ;
EXTERNAL    int     DBUGFLG8
#ifdef      INTRINIT
            = 0
#endif
            ;
EXTERNAL    FILE    *DBUGHNDL;
extern void DBUGBEGN();
extern void DBUGBIN();
extern void DBUGBOOL();
extern void DBUGCHAR();
extern void DBUGDBL();
extern void DBUGEND();
extern void DBUGFLOT();
extern void DBUGHEX();
extern void DBUGINT();
extern void DBUGLONG();
extern void DBUGOPEN();
extern void DBUGPTR();
extern void DBUGSTAK();
extern void DBUGSTR();
extern unsigned short int STACKPTR();

#endif
```

Listing 2. DEBUG.C file; Debugging functions

```
*  GENERAL DESCRIPTION:
*    Debug flags are defined in header "debug.h".
*    Debug flag values may be specified either in
*      environment or in a file depending upon #define below.
*      If set in environment:
*          DBUGFLGn=FALSE
*          DBUGFLGn=TRUE
*             for n = 0...8.
*      If environment variable does not exist, then
*         corresponding flag value is set to false.
*      If set in a file, in the first record give the string:
*          abcdefghi
*            a=0 for DBUGFLG0=FALSE or a=1 for DBUGFLG0=TRUE
*            ...
*          i=0 for DBUGFLG8=FALSE or h=1 for DBUGFLG8=TRUE
```

```
*      If flag does not exist or value is not 1, then
*        corresponding flag value is set to false. If file does
*        not exist, then all flag values are set to false.
*      At least one debug flag, DBUGFLG1 thru DBUGFLG8, must
*        be true for a debug output file to be produced.
*      DBUGFLG0 has follwing definition:
*          = FALSE  output is directed to disk file DEBUG.LST
*          = TRUE   output is directed to stderr.
*  SCCS IDENTIFICATION */
      static char SCCSID[] = "@(#)debug.c  5.3.8";
/*-----------------------------------------------------*/
/*  SPECIAL DEFINE */
#define  metachar int
#define  void     int
#define  string   char
#define  public
#define  local    static
#define  bool     int
#define  false    0
#define  true     1
#define  max(x,y)    ((x) >= (y) ? (x) : (y))
/*-----------------------------------------------------*/
#define     EXTERNAL extern
#include    <stdio.h>
#include    <dos.h>
#include    <time.h>
#undef      EXTERNAL
#define     EXTERNAL
#define     EXTRINIT
#include    <debug.h>
#undef      EXTRINIT
#undef      EXTERNAL
/*-----------------------------------------------------*/
#define  ENVFLAG     /* FILEFLAG if debug level flags are
                        in file. ENVFLAG if debug level
                        flags are in environment. */
#ifdef   FILEFLAG
local string FLAGNAME[]    = "debug.flg";
#endif

local string DBUGNAME[]    = "debug.lst";
local string PREVDAT[20+1] = "                    ";
/*-----------------------------------------------------*/
local void NEWTIME(P_CURDATE,P_CURTIME)
    string P_CURDATE[];         /*  out */
    string P_CURTIME[];         /*  out */

    /*  Get date in form "yyyy mmm dd www" and time in form
        hh:mm:ss" from SYSTIME, string returned by "asctime". */
{
    static string       FUNC[] = "NEWTIME";
    static string       D[20 + 1] = "                    ";
    static string       T[20 + 1] = "                    ";
    static string       SYSTIME[30+1];
    static string       DUMMY[3+1];

/*  begin */
    strcpy(SYSTIME,asctime(localtime(NULL)));
    strncpy(&D[ 0],&SYSTIME[20],4);   /*  "yyyy"      */
    strncpy(&D[ 4],&SYSTIME[ 3],8);   /*  " mmm dd " */
    strncpy(&D[12],&SYSTIME[ 0],3);   /*  "www"       */
    D[15] = '\0';                       /*  nul        */
    strncpy(&T[ 0],&SYSTIME[11],8);   /*  "hh:mm:ss" */
    T[ 8] = '\0';                       /*  nul        */
    strcpy(P_CURDATE,D);
    strcpy(P_CURTIME,T);
}
/*-----------------------------------------------------*/
local void DBUGTIME()
    /*  Display date if date has changed. Display time. */
{
    static string   FUNC[] = "DBUGTIME";
    string          CURDATE[20+1];
    string          CURTIME[20+1];

/* begin */
    NEWTIME(CURDATE,CURTIME);
    if (strncmp(CURDATE,PREVDAT,11) != 0) {
        strcpy(PREVDAT,CURDATE);
        fprintf(DBUGHNDL,"\n%s\n\n",CURDATE);
    }
    fprintf(DBUGHNDL,"%s ",CURTIME);
}
/*-----------------------------------------------------*/
local void DBUGXLCH(P_S,P_C)
```

Listing continues

This package can be tested by means of the example DEMODBUG.C (Listing 4). This example assumes a C compiler that allows a preprocessor variable to be defined as an argument on the compiler command line (DEBUG_1). Debugging code is not included unless DEBUG_1 is defined.

The debug flags used are given as follows:

Debug Flags	Description
DBUGFLG1	conditional debugging output from main program
DBUGFLG2	conditional debugging output from first function level
DBUGFLG3	conditional debugging output from second function level

The function call DBUGOPEN appears in "main" and assures that all debug flags are set according to the environmental variables, and will note that the debug output file is opened.

The flow trace is accomplished by the function call DBUGBEGN at the beginning of each function and the function call DBUGEND at the end of each function.

> *Additional debug calls may be used to display intermediate results, but a display of input parameters, output parameters and return values may be sufficient.*

The input parameters are displayed by the DBUGxxxx calls at the beginning of each function. The output parameters and the return values (if any) are displayed by the DBUGxxxx calls at the end of each function. Additional debug calls may be used to display intermediate results, but a display of input parameters, output parameters, and return values may be sufficient.

The debug output file DEBUG.LST is given in Listing 5. Note that all lines are time-stamped. This is of some use, but generally the debugging code takes more time than the rest of the code.

This program has been used with the Computer Innovations C86 Compiler Version 2.30f under MS-DOS/2.11 on a Lomas Lightning Computer (8086). With a suitable modification of the routines DBUGPTR and DBUGSTAK, it has also been used with a C compiler under UNIX/5 on the AT&T 3B2 Computer and also under UNIX on the Plexus P-40 Computer. □

Bill Rogers is a consultant in Princeton, New Jersey.

All the source code for articles in this issue is available on a single, MS-DOS disk. To order, send $14.95 to: Micro/Systems, *501 Galveston Drive, Redwood City, CA 94063, or call (415) 366-3600, ext. 216. Please specify the issue number. Source code is also available on CompuServe; type GO DDJ FORUM.*

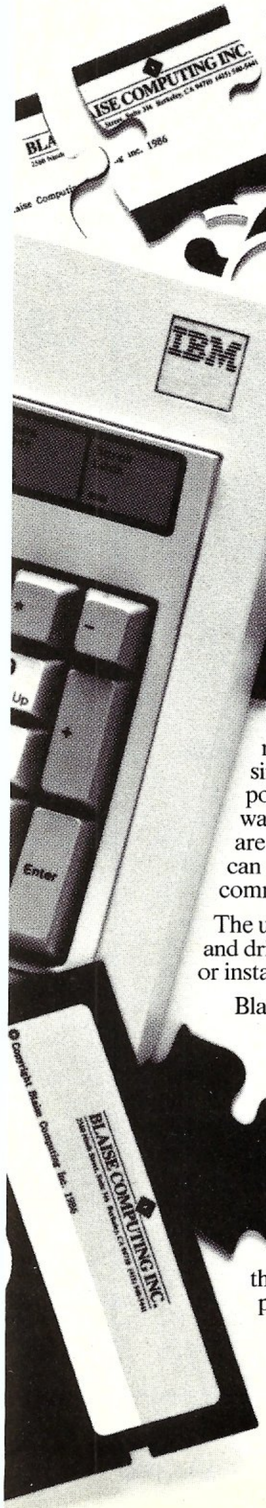
```
    string      P_S[];           /*  out. */
    metachar    P_C;             /*  in.  */
    /*  Translate a character so that it is "printable". */
{
    static string   FUNC[] = "DBUGXLCH";
    metachar        RTN;
    local char      LEADINCH = '^';
    local char      LTZEROCH = '!';
    local char      EOFCH    = '#';
    local char      BIGCH    = '*';
    local char      DELCH    = '?';

/* begin */
    P_S[0] = 'Z';
    P_S[1] = 'Z';
    P_S[2] = '\0';
    if         ((P_C ==   LEADINCH)) {
                                        /*  Double leadin. */
       P_S[0] = LEADINCH;
       P_S[1] = P_C;
       P_S[2] = '\0';
    } else if ((P_C <    0)                ) {
                               /*  Unknown character (< 0) */
       P_S[0] = LEADINCH;
       P_S[1] = LTZEROCH;
       P_S[2] = '\0';
    } else if ((P_C >=   0) && (P_C <=  31)) {
             /*  Control characters.  Exclude space here. */
       P_S[0] = LEADINCH;
       P_S[1] = P_C + '@';
       P_S[2] = '\0';
    } else if ((P_C >=  32) && (P_C <= 126)) {
              /* Printable characters. Include space here. */
       P_S[0] = P_C;
       P_S[1] = '\0';
    } else if (P_C == 127) {
                               /* Special handling for del. */
       P_S[0] = LEADINCH;
       P_S[1] = DELCH;
       P_S[2] = '\0';
    } else if (P_C == -1) {
                               /*  End of file "character". */
       P_S[0] = LEADINCH;
       P_S[1] = EOFCH;
       P_S[2] = '\0';
    } else {
                              /* Unknown character (> 127). */
       P_S[0] = LEADINCH;
       P_S[1] = BIGCH;
       P_S[2] = '\0';
    }
}   /*  end DBUGXLCH */
/*-----------------------------------------------------*/
local void DBUGXLST(P_TARGET,P_SOURCE)

    string  P_TARGET[];          /*  out. */
    string  P_SOURCE[];          /*  in.  */

    /*  Translate a string into a "printable" string. */
{
    static string   FUNC[] = "DBUGXLST";
    int             I;
    string          TEMP[2 + 1];

/* begin */
    I = 0;
    P_TARGET[0] = '\0';
    while (P_SOURCE[I] != '\0') {
        DBUGXLCH(TEMP,P_SOURCE[I]);
        strcat(P_TARGET,TEMP);
        I++;
    }
}   /*  end DBUGXLST */
/*------------------------------------------------------*/
public void DBUGBEGN(P_FUNC,P_SCCSID)

    string  P_FUNC[];            /*  in */
    string  P_SCCSID[];          /*  in */

    /*  Begin debug of a function P_FUNC with sccsid
        P_SCCSID. */
{
    static string   FUNC[] = "DBUGBEGN";

/* begin */
    if (! DBUGOPFL) {
        DBUGOPEN();
    }
    DBUGTIME();
    fprintf(DBUGHNDL,"%-8s: Begin with SCCSID='%s'\n",
        P_FUNC,P_SCCSID);
    fflush(DBUGHNDL);
}   /*  end DBUGBEGN */
```

Listing continues

Interlocking Pieces: Blaise and Turbo Pascal.

Now, for Turbo Pascal 4.0!

Whether you're a Turbo Pascal expert or a novice, you can benefit from using professional tools to enhance your programs. With Turbo POWER TOOLS PLUS™ and Turbo ASYNCH PLUS,™ Blaise Computing offers you all the right pieces to solve your 4.0 development puzzle.



Compiled units (TPU files) are provided so each package is ready to use with Turbo Pascal 4.0. Both POWER TOOLS PLUS and ASYNCH PLUS use units in a clear, consistent and effective way. If you are familiar with units, you will appreciate the organization. If you are just getting started, you will find the approach an illustration of how to construct and use units.

◆**POWER TOOLS PLUS** is a library of over 180 powerful functions and procedures like fast direct video access, general screen handling including multiple monitors, VGA and EGA 50-line and 43-line text mode, and full keyboard support, including the 101/102-key keyboard. Stackable and removable windows with optional borders, titles and cursor memory provide complete windowing capabilities. Horizontal, vertical, grid and Lotus-style menus can be easily incorporated into your programs using the menu management routines. You can create the same kind of moving pull down menus that Turbo Pascal 4.0 uses.

Control DOS memory allocation. Alter the Turbo Pascal heap size when your program executes. Execute any program from within your program and POWER TOOLS PLUS automatically compresses your heap memory if necessary. You can even force the output of the program into a window!

Write general interrupt service routines for either hardware or software interrupts. Blaise Computing's unique intervention code lets you develop memory resident (TSRs) applications that take full advantage of DOS capabilities. With simple procedure calls, "schedule" a Turbo Pascal procedure to execute either when pressing a "hot key" or at a specified time.

◆**ASYNCH PLUS** provides the crucial core of hardware interrupts needed to support asynchronous data communications. This package offers simultaneous buffered input and output to both COM ports, and up to four ports on PS/2 systems. Speeds to 19.2K baud, XON/XOFF protocol, hardware handshaking, XMODEM (with CRC) file transfer and modem control are all supported. ASYNCH PLUS provides text file device drivers so you can use standard "Readln" and "Writeln" calls and still exploit interrupt-driven communication.

The underlying functions of ASYNCH PLUS are carefully crafted in assembler and drive the hardware directly. Link these functions directly to your application or install them as memory resident.

Blaise Computing products include all source code that is efficiently crafted, readable and easy to modify. Accompanying each package is an indexed manual describing each procedure and function in detail with example code fragments. Many complete examples and useful utilities are included on the diskettes. The documentation, examples and source code reflect the attention to detail and commitment to technical support that have distinguished Blaise Computing over the years.

Designed explicitly for Turbo Pascal 4.0, Turbo POWER TOOLS PLUS and Turbo ASYNCH PLUS provide reliable, fast, professional routines—the right combination of pieces to put your Turbo Pascal puzzle together. **Complete price is $129.00 each.**

BLAISE COMPUTING INC.

2560 Ninth Street, Suite 316 Berkeley, CA 94710 (415) 540-5441

THE BLAISE MENU

Turbo POWER SCREEN *$129.00*
NEW! General screen management; paint screens; block mode data entry or field-by-field control with instant screen access. Now for Turbo Pascal 4.0, soon for C and BASIC.

Turbo C TOOLS *$129.00*
Full spectrum of general service utility functions including: windows; menus; memory resident applications; interrupt service routines; intervention code; and direct video access for fast screen handling. For Turbo C.

C TOOLS PLUS *$129.00*
Windows; menus; ISRs; intervention code; screen handling and EGA 43-line text mode support; direct screen access; DOS file handling and more. Specifically designed for Microsoft C 5.0 and QuickC.

ASYNCH MANAGER *$175.00*
Full featured interrupt driven support for the COM ports. I/O buffers up to 64K; XON/XOFF; up to 9600 baud; modem control and XMODEM file transfer. For Microsoft C and Turbo C or MS Pascal.

PASCAL TOOLS/TOOLS 2 *$175.00*
Expanded string and screen handling; graphics routines; memory management; general program control; DOS file support and more. For MS-Pascal.

KeyPilot *$49.95*
"Super-batch" program. Create batch files which can invoke programs and provide input to them; run any program unattended; create demonstration programs; analyze keyboard usage.

EXEC *$95.00*
NEW VERSION! Program chaining executive. Chain one program from another in different languages; specify common data areas; less than 2K of overhead.

RUNOFF *$49.95*
Text formatter for all programmers. Written in Turbo Pascal; flexible printer control; user-defined variables; index generation; and a general macro facility.

TO ORDER CALL TOLL FREE
800-333-8087

TELEX NUMBER - 338139

YES! Send me the right pieces!
Enclosed is $______ for ______ copies of ______.
☐ Please send me more information on your products.
CA residents add Sales Tax. Domestic orders add $4.00 for UPS shipping, $10.00 for Federal Express standard air.
Name: ______ Phone: (___) ______
Address: ______
City: ______ State: ______ Zip: ______
VISA or MC#: ______ Exp. Date: ______

Microsoft and QuickC are registered trademarks of Microsoft Corporation. Turbo Pascal is a registered trademark of Borland International.

```
/*---------------------------------------------------------*/
public void DBUGBIN(P_FUNC,P_NAME,P_VALUE)

    string          P_FUNC[];          /*  in */
    string          P_NAME[];          /*  in */
    unsigned int    P_VALUE;           /*  in */

    /*  Print a binary integer name P_NAME value P_VALUE
        in function P_FUNC. */
{
    static string   FUNC[] = "DBUGBIN";
    string          VALUE[16+1];
    unsigned int    MASK;
    int             I;

/* begin */
    if (! DBUGOPFL) {
        DBUGOPEN();
    }

    strcpy(VALUE,"");
    MASK = 0x8000;
    for (I = 0 ; I < 16 ; I++) {
        if (P_VALUE & MASK) {
            strcat(VALUE,"1");
        } else {
            strcat(VALUE,"0");
        }
        MASK = MASK >> 1;
    }
    DBUGTIME();
    fprintf(DBUGHNDL,"%-8s: %-16s=%-16sb\n",P_FUNC,P_NAME,VALUE);
    fflush(DBUGHNDL);
}   /*  end DBUGBIN */
/*---------------------------------------------------------*/
public void DBUGBOOL(P_FUNC,P_NAME,P_VALUE)

    string   P_FUNC[];          /*  in */
    string   P_NAME[];          /*  in */
    int      P_VALUE;           /*  in */

    /* Print an integer name P_NAME value P_VALUE in
       function P_FUNC. */
{
    static string   FUNC[] = "DBUGBOOL";
    string          TEMP[8];

/* begin */
    if (! DBUGOPFL) {
        DBUGOPEN();
    }
    if (P_VALUE) {
        strcpy(TEMP,"true");
    } else {
        strcpy(TEMP,"false");
    }
    DBUGTIME();
    fprintf(DBUGHNDL,"%-8s: %-16s=%s\n",P_FUNC,P_NAME,TEMP);
    fflush(DBUGHNDL);
}   /*  end DBUGBOOL */
/*---------------------------------------------------------*/
public void DBUGCHAR(P_FUNC,P_NAME,P_VALUE)

    string   P_FUNC[];          /*  in */
    string   P_NAME[];          /*  in */
    metachar P_VALUE;           /*  in */

    /* Print character name P_NAME value P_VALUE in
       function P_FUNC. */
{
    static string   FUNC[] = "DBUGCHAR";
    string          TEMP[2 + 1];

/* begin */
    if (! DBUGOPFL) {
        DBUGOPEN();
    }
    DBUGTIME();
    fprintf(DBUGHNDL,"%-8s: %-16s=",P_FUNC,P_NAME);
    DBUGXLCH(TEMP,P_VALUE);
    fprintf(DBUGHNDL,"'%s'\n",TEMP);
    fflush(DBUGHNDL);
}   /*  end DBUGCHAR */
/*---------------------------------------------------------*/
public void DBUGDBL(P_FUNC,P_NAME,P_VALUE)

    string     P_FUNC[];          /*  in */
    string     P_NAME[];          /*  in */
    long float P_VALUE;           /*  in */

    /* Print long float name P_NAME value P_VALUE in
       function P_FUNC.*/
{
    static string   FUNC[] = "DBUGDBL";

/* begin */
    if (! DBUGOPFL) {
        DBUGOPEN();
    }
    DBUGTIME();
    fprintf(DBUGHNDL,"%-8s: %-16s=%.15e\n",P_FUNC,P_NAME,P_VALUE);
    fflush(DBUGHNDL);
}   /*  end DBUGDBL */
/*---------------------------------------------------------*/
public void DBUGEND(P_FUNC)

    string   P_FUNC[];          /*  in */

    /*  End debugging a function P_FUNC. */
{
    static string   FUNC[] = "DBUGEND";

/* begin */
    if (! DBUGOPFL) {
        DBUGOPEN();
    }
    DBUGTIME();
    fprintf(DBUGHNDL,"%-8s: End\n",P_FUNC);
    fflush(DBUGHNDL);
}   /*  end DBUGEND */
/*---------------------------------------------------------*/
public void DBUGFLOT(P_FUNC,P_NAME,P_VALUE)

    string     P_FUNC[];          /*  in */
    string     P_NAME[];          /*  in */
    long float P_VALUE;           /*  in */

    /* Print float name P_NAME value P_VALUE in
       function P_FUNC. */
{
    static string   FUNC[] = "DBUGFLOT";

/* begin */
    if (! DBUGOPFL) {
        DBUGOPEN();
    }
    DBUGTIME();
    fprintf(DBUGHNDL,"%-8s: %-16s=%e\n",P_FUNC,P_NAME,P_VALUE);
    fflush(DBUGHNDL);
}   /*  end DBUGFLOT */
/*---------------------------------------------------------*/
public void DBUGHEX(P_FUNC,P_NAME,P_VALUE)

    string   P_FUNC[];          /*  in */
    string   P_NAME[];          /*  in */
    int      P_VALUE;           /*  in */

    /* Print hex integer name P_NAME value P_VALUE in
       function P_FUNC. */
{
    static string   FUNC[] = "DBUGHEX";

/* begin */
    if (! DBUGOPFL) {
        DBUGOPEN();
    }
    DBUGTIME();
    fprintf(DBUGHNDL,"%-8s: %-16s=%4xh\n",P_FUNC,P_NAME,P_VALUE);
    fflush(DBUGHNDL);
}   /*  end DBUGHEX */
/*---------------------------------------------------------*/
public void DBUGINT(P_FUNC,P_NAME,P_VALUE)

    string   P_FUNC[];          /*  in */
    string   P_NAME[];          /*  in */
    int      P_VALUE;           /*  in */

    /* Print integer name P_NAME value P_VALUE in
       function P_FUNC. */
{
    static string   FUNC[] = "DBUGINT";

/* begin */
    if (! DBUGOPFL) {
        DBUGOPEN();
    }
    DBUGTIME();
    fprintf(DBUGHNDL,"%-8s: %-16s=%6d\n",P_FUNC,P_NAME,P_VALUE);
    fflush(DBUGHNDL);
}   /*  end DBUGINT */
/*---------------------------------------------------------*/
public void DBUGLONG(P_FUNC,P_NAME,P_VALUE)

    string   P_FUNC[];          /*  in */
    string   P_NAME[];          /*  in */
    long int P_VALUE;           /*  in */
```

```
    /* Print long integer name P_NAME value P_VALUE in
       function P_FUNC. */
{
    static string   FUNC[] = "DBUGLONG";

/* begin */
    if (! DBUGOPFL) {
        DBUGOPEN();
    }
    DBUGTIME();
    fprintf(DBUGHNDL,"%-8s: %-16s=%11ld\n",P_FUNC,P_NAME,P_VALUE);
    fflush(DBUGHNDL);
}   /*  end DBUGLONG */
/*------------------------------------------------------*/
public void DBUGOPEN()

    /* open debug file. Debug file is automatically opened
       when ANY debug function called. */
{
    static string           FUNC[] = "DBUGOPEN";
    FILE                    *FLAGHNDL;
    unsigned char           *FLAGPTR;
    metachar                 C;
    string                   FLAGLIT[128];
    extern unsigned char    *envfind();
    extern char             *upper();

/* begin */
    if ( ! DBUGOPFL) {
        DBUGOPFL = true;

        /*  Set debug level flags from file */
#ifdef FILEFLAG
        FLAGHNDL = fopen(FLAGNAME,"r");
        if (FLAGHNDL != 0) {
            while (true) {
                C = fgetc(FLAGHNDL);
                if ( C != EOF) {
                    if (C == '1') {
                        DBUGFLG0 = true;
                    }
                } else {
                    break;
                }
                C = fgetc(FLAGHNDL);
                if ( C != EOF) {
                    if (C == '1') {
                        DBUGFLG1 = true;
                    }
                } else {
                    break;
                }
                C = fgetc(FLAGHNDL);
                if ( C != EOF) {
                    if (C == '1') {
                        DBUGFLG2 = true;
                    }
                } else {
                    break;
                }
                C = fgetc(FLAGHNDL);
                if ( C != EOF) {
                    if (C == '1') {
                        DBUGFLG3 = true;
                    }
                } else {
                    break;
                }
                C = fgetc(FLAGHNDL);
                if ( C != EOF) {
                    if (C == '1') {
                        DBUGFLG4 = true;
                    }
                } else {
                    break;
                }
                C = fgetc(FLAGHNDL);
                if ( C != EOF) {
                    if (C == '1') {
                        DBUGFLG5 = true;
                    }
                } else {
                    break;
                }
                C = fgetc(FLAGHNDL);
                if ( C != EOF) {
                    if (C == '1') {
```

Listing continues on page 54

Using MS-DOS Functions in C

by Mark Zeiger

Modularize your C program to make it portable between operating systems.

Most MS-DOS C compilers boast that they are "UNIX-compatible" and that their code can be ported to a UNIX system with "little or no change." However, I have found that some desirable features of UNIX cannot be emulated in a straightforward manner under MS-DOS.

There are also other functions that suffer in performance under MS-DOS if they are imitated. In UNIX, for example, the command processor expands wildcard file names. For instance, if there are four files in the current directory (*fn1, fn2, fn3,* and *fn4*), then calling a program *prog* *.* with the standard entry of *main(argc, argv)* will have *argc* = 5, *argv[0]* = *prog*, *argv[1]* = *fn1*, *argv[2]* = *fn2*, *argv[3]* = *fn3*, and *argv[4]* = *fn4*. Under MS-DOS *argc* would equal 2 and argv[1] would be *.*. The value of *argv[0]* would depend on the particular compiler being used, since MS-DOS versions earlier than 3.0 do not return the program name from the parameter line.

Also, many of the functions found in UNIX C compilers are not implemented in MS-DOS C compilers. This is because there is simply no equivalent UNIX system call in the MS-DOS environment. One example is the time and date functions of UNIX. MS-DOS does not return all the information about time zones, day of week, and other things that UNIX does (although some of the new DOS compilers do support such functions).

Further, control of screen and console functions is completely different under the two systems. For instance, if the user wishes to use the console input with no echo (writing a screen editor, for example), he or she must use function 6 under MS-DOS and IOCTL functions with UNIX.

The purpose of this article is to describe how MS-DOS function calls from a C program may be used. Using such function calls from a C program will make code non-portable between various operating systems. However, if the code is modularized well, then only certain sections will have to be changed to port an application from one system to another. Also, many utility programs (such as the one in this article) will probably never have to be ported to another system since either the other system already has the utility or the utility cannot be supported.

The program is called CHATT (CHange ATTributes, Listing 2). It can set or reset the system, hidden, and read/only attributes of files in MS-DOS. It is possible to change the attributes of many files in different directories with one command line:

```
chatt -s+r \*.com f*.asm subdir1
```

This will reset the system attribute and set the read/only attribute in all the .COM files in the root directory, all the .ASM files starting with *f* in the current directory, and, assuming *subdir1* is a subdirectory, all the files in the directory *subdir1*. To accomplish this in MS-DOS, you must use

EVEN MORE POWER AND FLEXIBILITY

BRIEF 2.0

Users and industry press alike have unanimously proclaimed BRIEF as the best program editor available today. Now, the best gets better, with the release of BRIEF 2.0.

Straight from the box, BRIEF offers an exceptional range of features. Many users find that BRIEF is the only editor they'll ever need, with features like real, multi-level Undo, flexible windowing and unlimited file size. But BRIEF has tremendous hidden power in its exclusive macro language. With it, you can turn BRIEF into your own custom editor containing the commands and features you desire. It's fast and easy.

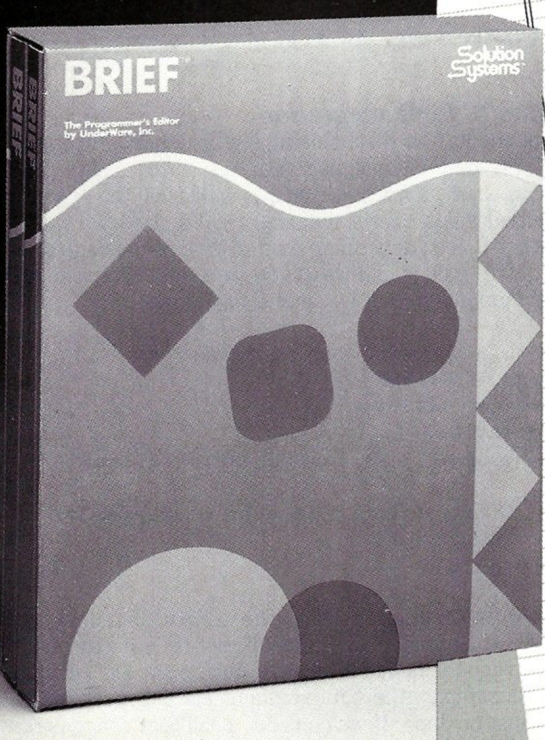
Jerry Pournelle, columnist for BYTE magazine summed it all up by saying BRIEF is, "**Recommended. If you need a general purpose PC programming editor, look no further.**" His point of view has been affirmed by rave reviews in C JOURNAL, COMPUTER LANGUAGE, DR. DOBB'S JOURNAL, DATA BASED ADVISOR, INFOWORLD AND PC MAGAZINE.

One user stated "BRIEF is one of the few pieces of software that I would dare call a masterpiece." Order BRIEF now and find out why. **BRIEF 2.0 is just $195. If you already own BRIEF, call for upgrade information.**

TO ORDER CALL: 1-800-821-2492
(in MA call 617-337-6963)

As always, BRIEF comes with a 30 day money-back satisfaction guarantee.

Solution Systems™
541 Main Street
Suite 410J
So. Weymouth, MA 02190
(617) 337-6963



Look at these BRIEF 2.0 enhancements!

Main Features:

- All new documentation with tutorials on basic editing, regular expressions **and** the BRIEF Macro Language.
- Setup program for easy installation and configuration. (Requires no knowledge of the macro language)
- Increased speed for sophisticated operations like Undo and Regular Expression Search.
- Expanded regular expressions, with matching over line boundaries.
- More block types, with marking by character, line or column.
- Command line editing (move cursor, add and delete characters, specify command parameters).
- Support for more programming languages.
- Optional borderless windows.
- Enhanced large display support, including wider displays.
- Reconfigurable indenting for C files (supports most indenting styles).

Plus the basic features that made BRIEF **SO** popular!

Basic Features:

- Full multi-level Undo
- Windows
- Edit many files at once
- File size limited only by disk space
- Automatic language sensitive indentation

Requires an IBM PC or compatible with at least 192K RAM.
BRIEF is a trademark of UnderWare, Inc.
Solution Systems is a trademark of Solution Systems.

the search first and search next functions, which are not usually implemented in standard libraries of MS-DOS C compilers. Also, some older compilers do not have functions that allow you to change the file mode (the more recent compilers do); therefore changing the attribute must be implemented as a DOS function call. And finally, some simple things, such as receiving a single character (without pressing RETURN and without having the character echo to the screen) are sometimes not implemented in many C libraries. The source listing of CHATT shows how to accomplish many of these tasks.

> *CHATT (CHange ATTributes) can set or reset the system, hidden, and read/only attributes of files in MS-DOS. It is possible to change the attributes of many files in different directories with one command line.*

Calling MS-DOS From C

The actual method of calling an MS-DOS function from C varies from implementation to implementation. Two examples that I have seen (which allow full control of all registers) are the following:

1. Microsoft, Lattice, Eco-88, and Turbo C Implementations

This method requires a pointer (the address) to two structures as well as the DOS function number. The structures are the register sets, which are unsigned integers for the 8086/88 and 80286. The state of the CPU flags is returned. The first structure contains the register values that MS-DOS expects. The second structure contains the register values set by MS-DOS upon return from the system call. The registers themselves are unsigned 16-bit values; therefore these calls are not only highly MS-DOS-dependent, they are also very CPU-dependent.

For example, to send a single character to the console under MS-DOS, the ASCII representation of the character is placed in the DL register and 6 is placed in the AH register. Then the *INT 21h* instruction is executed. The assembly language routine would be:

```
mov  dl,'A' ;print upper case "A"
mov  ah,6   ;raw console output
            ;function of DOS
int  21h    ;call DOS
```

The method of doing this in C would be:

```
/* define the structure */
struct REGS { unsigned ax,bx,cx,dx,si,di,es; };
unsigned flags;
struct REGS inr, outr; /* create two
                          structures  */
inr.dx = 'A'; /* 41 hex in DL */
inr.ax = 0x0600; /* 06 hex in AH, 00 in AL */
flags = sysint(0x21, &inr, &outr);
```

&inr is the address of the structure *inr*, which contains the values expected by the MS-DOS call; the structure *outr* will contain the values returned in the registers after the call (in this case nothing since the raw console output does not return anything).

As an example of a function that returns a value, consider the call to get a character from the console (raw console input). MS-DOS expects an 0ff in the DL register and 6 in the AH register if input is desired. It then returns either with the zero flag set if no character is ready or with the character in AL.

```
#define ZEROF 0x0001; /* zero flag is low
                         order bit of CPU
                         flags          */
struct REGS inr, outr;
unsigned flags;

do {
    inr.dx = 0x00ff;  /* console input
                         sub-function */
    inr.ax = 0x0600;
    flags = sysint(0x21, &inr, &outr);
} while ( (flags & ZEROF) == ZEROF);
```

Upon exit from the DO loop, *outr.ax* contains the character in the lower eight bits. To echo it, we can:

```
inr.dx = outr.ax & 0x00ff; /*clear DH
                             register.*/
inr.ax = 0x0600;
sysint(0x21, &inr, &outr); /* flags do not
                              matter here */
```

2. DeSmet C Implementation

This implementation is much simpler. It depends upon the external, pre-defined unsigned variables *_rax, _rbx, _rcx, _rdx, _rsi, _rdi, _res,* and *_rds,* as well as on the external character variables *_zerof* and *_carryf.* The last two will be set to either 0 or 1, depending on the state of the flags after the call. The call is *_doint(interrupt no).* Console I/O would be implemented in DeSmet C as follows:

```
do {          /* wait for character */
    _rdx = 0x00ff;
    _rax = 0x0600;
    _doint(0x21);
} while (_zerof); /* character is in _rax */

_rdx = _rax & 0x00ff; /* echo character */
_rax = 0x0600;
_doint(0x21);
```

One word of caution in using the *_doint()* function of DeSmet C. The variable *_rds* must be set to −1 if the value of the DS register is not to change during the DOS call. In the above examples the data segment has no meaning, so nothing must be done. If we consider a call that changes the disk transfer address (DTA) in MS-DOS, then we must be very careful with the data segment. Here, the correct call would be:

Listing 1. Executing a Microsoft-type system call

```
flags = sysint(function_number, &inr, &outr)

;assuming small memory model and C compiler
;parameters are pushed from left to right on stack
;segments $b$prog and $d$dataseg are unique to
;linking object modules using Eco-C88 Compiler.
;They will probably have to be changed for other
;compilers.

public _sysint

$d$dataseg segment
      inter_no dw 0,0 ;patched to pointer to interrupt..
$d$dataseg ends       ;..requested

$b$prog segment

_sysint proc near
     push bp      ;standard assembly language interface..
     mov  bp,sp   ;..to C programs

     ;stack is as follows       if small  if not small

     ; address of outr structure  bp+8      bp+10
     ; address of inr structure   bp+6      bp+8
     ; function number            bp+4      bp+6
     ; return address             bp+2      bp+2
     ; bp                         <--bp     bp

     ;if not small memory model, then return address
     ; would occupy 2 words

     ;it is not usually necessary to save registers
     ;other than BP when calling C functions.

     mov  ax,[bp+4] ;get MS-DOS function number in AL
     mov  ah,35h    ;MS-DOS get interrupt vector function
     int  21h           ;returns with vector in ES:BX
     mov  [inter_no],bx ;store vector in memory for later..
     mov  [inter_no+2],es  ;..far call

     mov  bx,[bp+6]    ;bx <- address of inr structure
     mov  ax,[bx]      ;ax <- inr.ax
     mov  cx,[bx+4]    ;cx <- inr.cx
     mov  dx,[bx+6]    ;dx <- inr.dx
     mov  si,[bx+8]    ;si <- inr.si
     mov  di,[bx+10]   ;di <- inr.di
     mov  es,[bx+12]   ;es <- inr.es
     mov  bx,[bx+2]    ;bx <- inr.bx

     pushf             ;this emulates an 8086 interrupt..
     call dword ptr [inter_no]  ;..which first pushes
                                ;flags and..
                                ;..then does far call

;Interrupt done. The pushf has been handled by IRET of
;interrupt procedure. Now put resulting registers in
;outr struct

     push bx
     mov  bx,[bp+8]   ;bx <- address of outr structure
     mov  [bx],ax     ;outr.ax <- ax
     mov  [bx+4],cx   ;outr.cx <- cx
     mov  [bx+6],dx   ;outr.dx <- dx
     mov  [bx+8],si   ;outr.si <- si
     mov  [bx+10],di  ;outr.di <- di
     mov  [bx+12],es  ;outr.es <- es

     pop  cx   ;now cx equals bx before bx was pushed
     mov  [bx+2],cx
     pushf            ;ax <- flags as return value
     pop  ax          ;return values in AX
     ret

_sysint endp
$b$prog   ends
       end
```

Listing 2. CHATT—A program to change the attributes of MS-DOS files

```
/* CHATT - change attribute.

   Changes file mode to or from READ/ONLY, SYSTEM, or HIDDEN
```

. . . listing continued

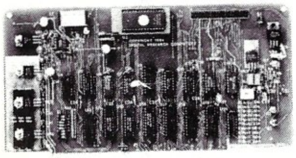
ALL SALES ARE MADE SUBJECT TO THE TERMS OF OUR 90 DAY LIMITED WARRANTY. A COPY OF THIS WARRANTY IS AVAILABLE FREE, ON REQUEST.

DIGITAL RESEARCH COMPUTERS

(214) 225-2309

S100 EPROM PROGRAMMER

OUR NEWEST DESIGN, FOR FAST EFFICIENT PROGRAMMING OF THE MOST POPULAR EPROM'S ON YOUR S100 MACHINE. COMES WITH MENU DRIVEN SOFTWARE THAT RUNS UNDER CP/M 2.2 (8 INCH). PC BOARD SET CONSISTS OF (S100) MAIN LOGIC BOARD REMOTE PROGRAMMING CARD AND SIX PERSONALITY MINI BOARDS FOR 2716, 2532, 2732, 2732A, 2764, AND 27128. SOLD AS BARE PC BOARD SET ONLY WITH FULL DOC. SOFTWARE FEATURES "FAST" PROGRAMMING ALGORITHM. FOR Z80 BASED SYSTEMS.

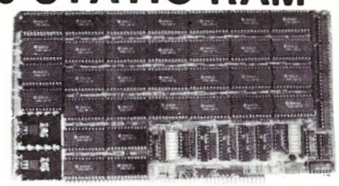


PC BOARD SET, FULL DOCUMENTATION, 8 IN. DISKETTE WITH SOFTWARE.

PRICE CUT! $39.95

64K S100 STATIC RAM

$99.00 KIT



LOW POWER!

150 NS ADD $10

BLANK PC BOARD WITH DOCUMENTATION $49.95

SUPPORT ICs + CAPS $17.50

FULL SOCKET SET $14.50

FULLY SUPPORTS THE NEW IEEE 696 S100 STANDARD (AS PROPOSED)

ASSEMBLED AND TESTED ADD $50

FEATURES: ***PRICE CUT!***

* Uses new 2K x 8 (TMM 2016 or HM 6116) RAMs.
* Fully supports IEEE 696 24 BIT Extended Addressing.
* 64K draws only approximately 500 MA.
* 200 NS RAMs are standard. (TOSHIBA makes TMM 2016s as fast as 100 NS. FOR YOUR HIGH SPEED APPLICATIONS.)
* SUPPORTS PHANTOM (BOTH LOWER 32K AND ENTIRE BOARD).
* 2716 EPROMs may be installed in any of top 48K.
* Any of the top 8K (E000 H AND ABOVE) may be disabled to provide windows to eliminate any possible conflicts with your system monitor, disk controller, etc.
* Perfect for small systems since BOTH RAM and EPROM may co-exist on the same board.
* BOARD may be partially populated as 56K.

128K S100 STATIC RAM/EPROM BOARD

JUST OUT! USES POPULAR 8K X 8 STATIC RAMS (6264) OR 2764 EPROMS. FOR 8 OR 16 BIT DATA TRANSFERS! IEEE 696 STANDARD. LOW POWER. KITS ARE FULLY SOCKETED. FULL DOC AND SCHEMATICS INCLUDED. 24 BIT ADDRESSING.

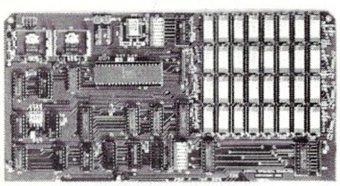
NEW!

$59.95	$219.00	$139.00
BARE PC BOARD	128K RAM KIT	128 EPROM KIT

256K S-100 SOLID STATE DISK SIMULATOR!

WE CALL THIS BOARD THE "LIGHT-SPEED-100" BECAUSE IT OFFERS AN ASTOUNDING INCREASE IN YOUR COMPUTER'S PERFORMANCE WHEN COMPARED TO A MECHANICAL FLOPPY DISK DRIVE.

PRICE CUT!



FEATURES:

* 256K on board, using +5V 64K DRAMS.
* Uses new Intel 8203-1 LSI Memory Controller.
* Requires only 4 Dip Switch Selectable I/O Ports.
* Runs on 8080 or Z80 S100 machines.
* Up to 8 LS-100 boards can be run together for 2 Meg. of On Line Solid State Disk Storage.
* Provisions for Battery back-up.
* Software to mate the LS-100 to your CP/M* 2.2 DOS is supplied.
* The LS-100 provides an increase in speed of up to 7 to 10 times on Disk Intensive Software.
* Compare our price! You could pay up to 3 times as much for similar boards.

BLANK PCB (WITH CP/M* 2.2 PATCHES AND INSTALL PROGRAM ON DISKETTE) $24.95 (8203-1 INTEL $29.95)

CLOSE OUT! **BLANK PCB ONLY:** $24.95

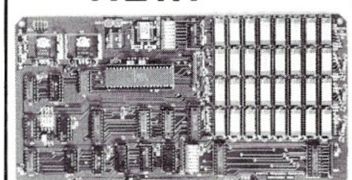
#LS-100

1 MEG. S-100 SOLID STATE DISK SIMULATOR!

WE CALL THIS BOARD THE "LIGHT-SPEED-100" BECAUSE IT OFFERS AN ASTOUNDING INCREASE IN YOUR COMPUTER'S PERFORMANCE WHEN COMPARED TO A MECHANICAL FLOPPY DISK DRIVE.

LS 100 II

NEW!



FEATURES:

* 1 Meg. on board, using +5V 256K DRAMS. (With Parity)
* Uses new Intel 8203-1 LSI Memory Controller.
* Requires only 4 Dip Switch Selectable I/O Ports.
* Runs on 8080 or Z80 S100 machines.
* Up to 4 LS-100 boards can be run together for 4 Megs. of On Line Solid State Disk Storage.
* Provisions for Battery back-up.
* Software to mate the LS-100 to your CP/M* 2.2 DOS is supplied.
* The LS-100 provides an increase in speed of up to 7 to 10 times on Disk Intensive Software.
* Compare our price! You could pay up to 3 times as much for similar boards.

BLANK PCB (WITH CP/M* 2.2 PATCHES AND INSTALL PROGRAM ON DISKETTE) $59.95 (8203 1 INTEL $29.95)

(ADD $50 FOR A&T) $259.00

#LS-100 II (FULL 1 M.B. KIT)

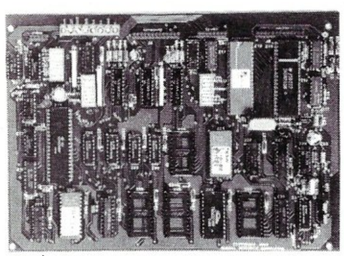
1 MEGA BYTE!

ZRT-80 CRT TERMINAL BOARD!

A LOW COST Z-80 BASED SINGLE BOARD THAT ONLY NEEDS AN ASCII KEYBOARD, POWER SUPPLY, AND VIDEO MONITOR TO MAKE A COMPLETE CRT TERMINAL. USE AS A COMPUTER CONSOLE, OR WITH A MODEM FOR USE WITH ANY OF THE PHONE-LINE COMPUTER SERVICES.

FEATURES:

* Uses a Z80A and 6845 CRT Controller for powerful video capabilities.
* RS232 at 16 BAUD Rates from 75 to 19,200.
* 24 x 80 standard format (60 Hz).
* Optional formats from 24 x 80 (50 Hz) to 64 lines x 96 characters (60 Hz).
* Higher density formats require up to 3 additional 2K x 8 6116 RAMS.
* Uses N.S. INS 8250 BAUD Rate Gen. and USART combo IC.
* 3 Terminal Emulation Modes which are Dip Switch selectable. These include the LSI-ADM3A, the Heath H-19, and the Beehive.
* Composite or Split Video.
* Any polarity of video or sync.
* Inverse Video Capability.
* Small Size: 6.5 x 9 inches.
* Upper & lower case with descenders.
* 7 x 9 Character Matrix.
* Requires Par. ASCII keyboard.



$89.95 #ZRT-80 A&T ADD $50

(COMPLETE KIT, 2K VIDEO RAM)

FOR 8 IN. SOURCE DISK OR PC-XT FORMAT 5¼ IN. ADD $10

OUR BEST SELLER!

THE NEW 65/9028 VT ANSI VIDEO TERMINAL BOARD!

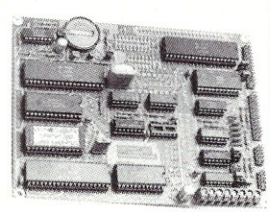
★ FROM LINGER ENTERPRISES ★

A second generation, low cost, high performance, mini sized, single board for making your own RS232 Video Terminal. This highly versatile board can be used as a stand alone video terminal, or without a keyboard, as a video console. VT100, VT52 Compatible.

FEATURES:

* Uses the new CRT9128 Video Controller driven by a 6502A CPU
* On-Screen Non-Volatile Configuration
* 10 Terminal Modes: ANSI, H19, ADM-5, WYSE 50, TVI-920, KT-7, HAZ-1500, ADDS 60, QUME-101, and Datapoint 8200
* Supports IBM PC/XT, and Parallel ASCII Keyboards
* Supports standard 15.75 kHz (Horiz.)
* Composite or Split Video (50/60 Hz)
* 25 X 80 Format with Non-Scrolling User Row
* Jump or Smooth Scroll
* RS-232 at 16 Baud Rates from 50 to 19,200
* On Board Printer Port
* Wide and Thin Line Graphics
* Normal and Reverse Screen Attributes
* Cumulative Character Attributes: De-Inten, Reverse, Underline and Blank
* 10 Programmable Function Keys and Answerback message
* 5 X 8 Character Matrix or 7 X 9 for IBM Monitors
* Mini Size: 6.5 X 5 inches
* Low Power: 5VDC @ .7A, ± 12VDC @ 20mA.

MICRO SIZE!



$79.95 FULL KIT

w/100 Page Manual

ADD $40 FOR A&T

OPTIONAL EPROM FOR PC/XT STYLE SERIAL KEYBOARD: $15

SOURCE DISKETTE: PC/XT FORMAT 5¼ IN. $15

Digital Research Computers

P.O. BOX 381450 • DUNCANVILLE, TX 75138 • (214) 225-2309

TERMS: Add $3.00 postage. Orders under $15 add 75¢ handling. No C.O.D. We accept Visa and MasterCard. Tex. Res. add 6-1/4% Tax. Foreign orders (except Canada) add 20% P & H. Orders over $50 add 85¢ for insurance.

*TM OF DIGITAL RESEARCH INC. (CALIF.)

WE ARE NOT ASSOCIATED WITH DIGITAL RESEARCH INC. (CALIF.) THE SUPPLIERS OF CPM SOFTWARE

```
char dta[128];  /* define 128 byte area */
_rdx = (unsigned) &dta[0]; /* cast pointer to
                                        unsigned */
_rax = 0x1a00; /* set DTA function */
_rds = -1; /* assume small memory... */
_doint(0x21); /* ..model by not
                    changing DS */
```

If *_rds* is not set to −1 or assigned a value, then it will be assigned the value 0. Thus the disk transfer area would have been set to *0000:(address of dta[0])*, probably wiping out the entire operating system.

Suppose that your C implementation does not implement either of these methods but you wish to use one of them. The first method is easier to code. Listing 1 offers assembly language code that executes the Microsoft-type system call.

Using MS-DOS within C Applications

Now that the calling conventions have been discussed, we can examine the program CHATT.C to see how MS-DOS functions can be used within C programs. The entire logic of the program lies in the FOR loop that starts on line 160 of Listing 2. However, before entering the loop, two items must be considered.

First, since the program either accepts ***argv[1]*** as a control string or as a file name, we must test ***argv[1]*** to see which it is. If the argument starts with a + or −, it is assumed to be a control string specifying which attributes to set or reset; otherwise, it is assumed to be a file name (if no control string is specified, then attributes will be displayed only). Therefore, lines 140–150 test the first character of the first argument for + or − and set appropriate flags depending on what is found. If there is a control argument,

```
0004:   interactively by first displaying current mode and then
0005:   asking if you wish a change to be made. Wildcards may be
0006:   used.
0007:
0008:    Created by Mark Zeiger
0009:    Written in Eco-C88
0010: */
0011:
0012: /* Command line parameters
0013:
0014:   chatt [control] [path\]filename [[path\]filename ...]
0015:
0016:   "Control" parameter specifies whether attribute should
0017:   be turned on or off and whether you should be asked if
0018:    change is to be done.
0019:
0020:   R specifies READ/ONLY, H specifies HIDDEN, and S specifies
0021:   SYSTEM.  If path is not specified, default directory is
0022:   used.  Filename may include wildcard characters. A direc-
0023:   tory alone is expanded to all files in directory (i.e.
0024:   test = test\*.* if test is a directory).
0025:
0026:   An "N" in control parameter will allow changes to be make
0027:   without asking. If no control parameter is specified then
0028:   attributes will be displayed only.
0029:
0030:   The control parameter takes the form of the string
0031:   +/-R | +/-H | +/-S | N in any order, except that "N" must
0032:   not come first (the control string must start with a + or
0033:   -).
0034:       example:
0035:
0036:   chatt +r-h+sn abc\*.com changes all COM files in directory
0037:                           "abc" to READ/ONLY and SYSTEM and
0038:                           takes away HIDDEN attribute.  No
0039:                           prompt is issued  for each file.
0040:
0041:   chatt -s+h abc\*.* d:*.asm changes all files in directory "abc"
0042:                              to HIDDEN and all ASM files in current
0043:                              directory of drive "D" to HIDDEN. Also
0044:                              takes away SYSTEM attribute of those
0045:                              files. You will be prompted if change
0046:                              is to be done.
0047:
0048:   chatt -s+h abc d:*.asm     same as above.
0049:
0050:   chatt \abc\*.com           lists attributes of all COM files
0051:                              in directory \abc.
0052:
0053: */
0054:
0055: #define TRUE  1
0056: #define FALSE 0
0057: #define NULL  0
0058:
0059: /* DOS call parameters */
0060:
0061: #define MSDOS   0x0021     /* MS-DOS interrupt */
0062: #define SETMODE 0x4301     /* subfunction number in AL */
0063: #define GETMODE 0x4300     /* subfunction number in AL */
0064: #define SEARCHF 0x4e00     /* search for first occurance */
0065: #define SEARCHN 0x4f00     /* search for next occurances */
0066: #define SETDMA  0x1a00     /* set DMA address */
0067: #define CONIO   0x0600     /* raw console I/O */
0068: #define CARRYF  0x0001     /* position of carry flag */
0069: #define ZEROF   0x0040     /* position of zero flag */
0070:
0071: /* file attribute bits for function 43H */
0072:
0073: #define RO     0x0001
0074: #define HIDDEN 0x0002
0075: #define SYSTEM 0x0004
0076: #define VOLUME 0x0008
0077: #define SUBDIR 0x0010
0078:
0079: /* errors from setmode and search commands */
0080:
0081: #define NOFILE   0x0002    /* file not found    */
0082: #define NOPATH   0x0003    /* path not found    */
0083: #define NOACCESS 0x0005    /* access denied     */
0084: #define NOFILES  0x0012    /* no more files on SEARCHN */
0085:
0086: /* BIOS interrupt 10H constants */
0087:
0088: #define CURR_VIDEO  0x0f00
0089: #define READ_CURSOR 0x0300
0090: #define SET_CURSOR  0x0200
0091: #define VIDEO_INT   0x0010
0092:
0093: /* Structure filled in by SEARCHF command of MSDOS */
0094:
0095: struct DMA {
0096:           char     reserved[21];  /* reserved by MS-DOS */
0097:           char     attribute;     /* attribute of file */
0098:           unsigned time;
0099:           unsigned date;
0100:           unsigned size_L;      /* file size - low word */
0101:           unsigned size_h;      /* file size - high word */
0102:           char     fname[13];  /* parsed name of file   */
0103:           };
0104:
0105:
0106: /* Structures used by Eco-C88 for interrupt calls */
0107:
0108: struct REGS { unsigned ax,bx,cx,dx,si,di,es; };
0109:
0110: struct REGS inr, outr;
0111:
0112: /*---------------------- MAIN ------------------------ */
0113:
0114: main(argc, argv)
0115:
0116: int     argc;
0117: char    *argv[];
0118:
0119: {
0120: char    atton_off,    /* is it a + or - */
0121:         path[64],     /* holds path name up to 64 bytes */
0122:         new_arg[64]; /* might hold subdir concat with \*.* */
0123: int     bslpos;       /* position of last backslach in path name */
0124: struct  DMA dmabuf;  /* used for MS-DOS search function */
0125: int     i;            /* used to count parameters */
0126: int     carryf;       /* carry flag for MS-DOS calls */
0127:
0128: void     fpe(), bad_syntax(), pfname(),
0129:          set_att_masks(), change_att(),
0130:          concat();
0131: char     lastchar();
0132: int      getpath();
0133: unsigned att_on_mask, att_off_mask; /* masks to set/reset attributes */
0134: unsigned ask;                        /* prompt for each change ???    */
0135: int      start_file;          /* which argv[] contains 1st path\file */
0136: int      display_att_only;    /* flag to display attribs only */
0137:
0138:     if (argc < 2) bad_syntax();
0139:
0140:     atton_off = argv[1][0];
0141:     if (atton_off != '+' && atton_off != '-') {
0142:        start_file = 1;                  /* argv[1] is a path */
0143:        display_att_only = TRUE;
0144:     }
0145:     else {
```

. . . listing continued

the function *set_att_masks* is called. These masks are used later to set or reset file attributes.

The other task that must be performed is to set the MS-DOS disk transfer area for the *search first* and *search next* functions. The *search first* function fills in the DTA (lines 95–104). Therefore MS-DOS function *1A hex* is called with the address of the desired memory location in the DX register (note that the cast from pointer to unsigned is necessary with many of the C compilers—this cast is CPU-dependent).

Line 156 seems to be necessary with some calls and not with others. It simply puts the current value of the DS register in the extra segment (*getds()* is an Eco-C88 function that returns the value of DS). Although none of the MS-DOS calls used in the program document the use of the ES register, if the function is not performed, the program does not work. Another option could be to rewrite the *_sysint()* function so that *inr.es* is not used.

> *The search functions require a full file name although some DOS utilities require only a subdirectory or even a drive specification.*

We can now enter the main program loop, starting with either *argv[1]* or *argv[2]* depending on whether *argv[1]* is a file name. The program searches for all files in a particular path that satisfy the wildcard conditions.

The *search* functions require a full file name, although some DOS utilities require only a subdirectory or even a drive specification. A common example is the *dir a:* command which DOS interprets as *dir a:*.**. Also, a command such as *dir abc,* where *abc* is a subdirectory, is changed to *dir abc*.** by DOS. Unfortunately, the *search* functions require a full file name and will not make the above expansions; these must be done in the program. Therefore we must test the name for three things:

1. If the last character is a colon, we append the *.* characters to the file name (lines 167–170). Therefore *A:* becomes *A:*.**.
2. If the last character is a \, then the name must be a subdirectory and we again append *.* (lines 171–174).
3. If the name is a subdirectory (tested with the *get file attribute* function of MS-DOS), then we concatenate a *.* to the path name (lines 175–183).

Note that the *get attribute* function is called with *43 hex* in AH. AL specifies whether to get or set the attribute (0 = get, 1 = set). Also, DS:DX points to the address of the ASCIIZ file name. "ASCIIZ" is an abbreviation for an ASCII string ending with a byte of zeros.

Now that *argv[i]* points to the full name, we are ready to use the *search* functions. There is only one problem. The *search* functions fill the structure in the DTA with the first and subsequent names, but these names are only file names without the path. To change the attribute we are going to

```
0146:       start_file = 2;               /* argv[1] is control parameter */
0147:       display_att_only = FALSE;     /* and argv[2] starts files */
0148:       if (argc < 3) bad_syntax();  /* must have at least 1 file */
0149:       set_att_masks(argv[1], &att_on_mask, &att_off_mask, &ask);
0150:    }
0151:
0152:    /* set the MS-DOS DMA x'fer address to dmabuf */
0153:
0154:    inr.ax = SETDMA;
0155:    inr.dx = (unsigned) &dmabuf;
0156:    inr.es = getds();
0157:    sysint(MSDOS, &inr, &outr);
0158:
0159:
0160:    for (i = start_file; i < argc; i++) {
0161:
0162:       /* Changes a subdirectory name to subdir\*.*
0163:          or changes root to full search  (\ -> \*.*)
0164:          or changes drivespec to drivespec:\*.* (c: - > c:\*.*)
0165:       */
0166:
0167:       if (lastchar(argv[i]) == ':') {
0168:          concat(new_arg, argv[i], "*.*");
0169:          argv[i] = new_arg;
0170:       }
0171:       else if (lastchar(argv[i]) == '\\') {
0172:          concat(new_arg, argv[i], "*.*");
0173:          argv[i] = new_arg;
0174:       }
0175:       else {
0176:          inr.ax = GETMODE;
0177:          inr.dx = (unsigned) argv[i];
0178:          carryf = int86(MSDOS, &inr, &outr);
0179:          if (outr.cx == SUBDIR) {
0180:             concat(new_arg, argv[i], "\\*.*");
0181:             argv[i] = new_arg;
0182:          }
0183:       }
0184:
0185:       bslpos = getpath(argv[i],path);
0186:
0187:       fpe("\n===================> ");
0188:       fpe(argv[i]);
0189:       fpe(" <===================\n");
0190:
0191:
0192:      inr.ax = SEARCHF;
0193:      inr.dx = (unsigned) argv[i];
0194:      inr.es = getds();
0195:      inr.cx = 0x001f;        /* search for file with any attri- */
0196:      carryf = sysint(MSDOS, &inr, &outr); /* bute except archive */
0197:
0198:       if ( (carryf & CARRYF) == CARRYF) {
0199:           if ( (outr.ax == NOFILES) || (outr.ax == NOPATH) ) {
0200:               fpe("\n");
0201:               fpe(argv[i]);
0202:               fpe(" not found\n\n\r");
0203:           }
0204:           else {
0205:               fpe("\nUndefined error\n\007");
0206:               exit(1);
0207:           }
0208:       }
0209:
0210:       while ( (carryf & CARRYF) != CARRYF) {
0211:        pfname(path, &(dmabuf.fname[0]), bslpos);
0212:
0213:        fpe(path);
0214:        display_type(dmabuf.attribute);
0215:        if ( (display_att_only != TRUE) && (dmabuf.attribute != SUBDIR))
0216:            change_att(path, dmabuf.attribute, att_on_mask,
0217:                                                 att_off_mask, ask);
0218:        else fpe("\n");
0219:
0220:
0221:         inr.ax = SEARCHN;
0222:         inr.es = getds();
0223:         carryf = sysint(MSDOS, &inr, &outr);
0224:       }
0225:    } /* end of for loop */
0226:
0227:    exit(0);
0228: }
0229:
0230:
0231: /*---------------------- SET_ATT_MASKS ----------------------*/
0232:
0233: /* When it comes time to set or reset attributes, directory
0234:    attribute byte will be "or"ed with "att_on_mask" to set
0235:    attribute bits we want on and then "and"ed with "att_off_mask"
0236:    to reset attributes we want off. This is done in function
0237:    "change_att()".
0238: */
```

```c
void set_att_masks(att_string, att_on_mask, att_off_mask, ask)
char *att_string;
unsigned *att_on_mask, *att_off_mask, *ask;
{
int sub_str();
        *att_on_mask = 0;
        *att_off_mask = 0xffff;
        if (sub_str(att_string, "+R")) *att_on_mask |= RO;
        else if (sub_str(att_string, "-R")) *att_off_mask &= RO;
        if (sub_str(att_string, "+H")) *att_on_mask |= HIDDEN;
        else if (sub_str(att_string, "-H")) *att_off_mask &= HIDDEN;
        if (sub_str(att_string, "+S")) *att_on_mask |= SYSTEM;
        else if (sub_str(att_string, "-S")) *att_off_mask &= SYSTEM;
        if (sub_str(att_string, "N")) *ask = FALSE;
        else *ask = TRUE;
}
/*---------------------- SUB_STR --------------------------*/
/* Returns beginning position of second string in first string
   or zero if second string is not a substring of first. Case
    insensitive. */
int sub_str(main_string, sub_string)
char *main_string, *sub_string;
{
int     pos, len_main, len_sub;
        len_main = strlen(main_string);
        len_sub = strlen(sub_string);
        if (len_main < len_sub) return 0;
        for (pos = 0; pos < len_main-len_sub+1; pos++)
          if (xstrcomp(main_string, sub_string,pos,len_sub)) return pos+1;
        return 0;
}
/*-------------------- XSTRCOMP ---------------------------*/
/* Compares "sub_string" substring in "main_string" starting
   at "start_pos" with a length of "length". Case insensitive.
*/
int xstrcomp(main_string, sub_string, start_pos, length)
char *main_string, *sub_string;
int  start_pos, length;
{
int i;
    for (i = start_pos; i < start_pos+length; i++)
       if (toupper(*(main_string+i)) != toupper(*sub_string++)) return 0;
    return 1;
}
/*---------------------- CONCAT ------------------------------*/
/* Concatenates "first" string followed by "second" and puts result
   in "new" string.
*/
void concat(new, first, second)
char *new, *first, *second;
{
        while (*first != (char) NULL) *new++ = *first++;
        while (*second != (char) NULL) *new++ = *second++;
        *new = (char) NULL;
}
/*---------------------- LASTCHAR ------------------------------*/
/* Returns the last (non-null) character of a string
*/
char lastchar(string)
```

. . . listing continued

need the full path name. Thus we need the function *getpath()*, which is called in line 185. This function does two things: (1) It copies the path name (without the file name) into a character array called *path[ ]*, and (2) it then returns the position of the last *backslash* in this array. Thus, at a later time, the file name from each *search* call can be concatenated to the path name. If there is no path name, then *getpath()* returns a −1.

Finally, we can use the *search first* function to find the first file that matches our file specification. Naturally, AH contains the function number (*4E hex* in this case) and DS:DX points to the full path name. Also, the CX register contains the attribute mask; only those files whose attributes match the set bits will be found. These bits are defined in lines 73–77. If a matching file is found, the carry flag will be reset.

As long as the carry flag is not set, we will continue performing the functions *pfname()* and possibly *change_att()*. Function *pfname()* takes the file name from the structure filled in by *search* and appends it to the path name in *path[ ]*. We can then do what we wish with this file in *path[ ]* (in this case displaying and possibly changing the attribute).

After we are done with the first file, we search for additional files using the *search next* function. Note that this function needs no parameters—it depends upon the information in the DTA filled by the *search first* call. The program keeps searching and filling in the path name until no more files are found (carry set). Once finished, it goes on to examine the next *argv[i]*.

Almost any program manipulating multiple files can be implemented in this manner. The body of the program is really lines 213–218 (along with line 149 and some of the *define* statements). I have written a program that will print multiple files by just replacing lines 213–218 with a call to a print routine.

Using Other Interrupts

The *sysint()* call has been used for MS-DOS interrupts and can be used for other interrupts as well. I have used it in this program to print strings since the C function *fputs()* is incredibly slow (see the functions *fpe()* and *echo()*). One other important example is using interrupt 10H, the IBM screen-handling interrupt. This allows you to set the cursor position, print characters with attributes such as blink and reverse, and even do simple graphics. An example of using interrupt 10H is shown in the function *cursor_col()*.

To set the cursor, we must first get the current active page. Calling INT 10H with AH = 15 (decimal) will return with the BH register containing the current active page. The page number is saved and then we get the current cursor position. I do this because I want to change the column only and leave the row as is. With AH = 3 and BH = desired page, INT 10H will return the cursor position row in DH and the column in DL. Line 555 then changes DL only and calls INT 10H with the new cursor position in the DX register and the desired page in the BH register.

I hope this discussion has shown you how to use MS-DOS calls in general and how to use them from C in particular. □

Mark Zeiger is a product manager for Osicom Technology, an OEM for computer products based in Rockaway, New Jersey.

All the source code for articles published in Micro/Systems *is available on an MS-DOS disk. To order, send $14.95 to* Micro/Systems, *501 Galveston Drive, Redwood City, CA 94063; or call Tim at (415) 366-3600. Please specify the issue number. Source code is also available on CompuServe; type GO DDJFORUM.*

```
0332: char *string;
0333:
0334: {
0335:
0336:         return string[strlen(string)];
0337: }
0338:
0339:
0340: /*---------------------- GETPATH -----------------------*/
0341:
0342: /* Puts path name (without file name) in array "path" and returns
0343:    position of last backslash. If no path name (default directory)
0344:    then puts NULL string in path[] and sets bslpos to - 1. A path
0345:    is also considered to be a drive specification without a
0346:    subdirectory; hence the test for :.
0347: */
0348:
0349: int getpath(argvs,path)
0350:
0351: char *argvs, *path;
0352:
0353: {
0354: static char ch;
0355: register int  i;
0356: int bslpos = -1;
0357:
0358:    for (i = 0; (ch = argvs[i]) != (char) NULL; i++)
0359:        if (ch == '\\' || ch == ':') bslpos = i;
0360:
0361:    if (bslpos != -1) for (i = 0; i <= bslpos; i++) path[i] = argvs[i];
0362:    return bslpos;
0363: }
0364:
0365: /*----------------------- PFNAME --------------------------*/
0366:
0367: /* Puts file name (from dmabuf.fname) at end of path (which contains
0368:    pathname). bslpos is position of last backslash .
0369: */
0370:
0371: void pfname(path,fname,bslpos)
0372:
0373: char *path, *fname;
0374: int bslpos;
0375:
0376: {
0377: register i, j;
0378: static char ch;
0379: void display_type();
0380:
0381:   j = bslpos != -1  ? bslpos+1 : 0;
0382:   for (i = 0; (ch = fname[i]) != (char) NULL; i++, j++) path[j] = fname[i];
0383:   path[j] = (char) NULL;
0384: }
0385:
0386:
0387: /*--------------------- CHANGE_ATT --------------------------*/
0388:
0389: /* Changes attribute of file after query (unless "ask" is FALSE).
0390:    No change if subdirectory
0391: */
0392:
0393: void change_att(fullname, att, att_on_mask, att_off_mask, ask)
0394:
0395: char *fullname, att, att_on_mask, att_off_mask;
0396: int ask;
0397:
0398: {
0399: static char answer;
0400: char input();
0401: unsigned mode_type();
0402: void echo(), fpe();
0403: static int carryf;
0404:
0405:     if (ask == TRUE) {
0406:         fpe("    Change? ");
0407:         do {
0408:            answer = input();
0409:            if (answer == 3) exit(2);    /* abort if ^C */
0410:         }
0411:         while ((toupper(answer) != 'Y') && (toupper(answer) != 'N'));
0412:
0413:         echo(answer);
0414:     }
0415:     else answer = 'Y';            /* force change without asking */
0416:
0417:     if (toupper(answer) == 'Y') {
0418:         inr.ax = SETMODE;
0419:         inr.cx = att | att_on_mask;
0420:         inr.cx = (inr.cx & att_off_mask) & 0x00ff;
0421:         inr.dx = (unsigned) fullname;
0422:         carryf = sysint(MSDOS, &inr, &outr);
0423:         if ( (carryf & CARRYF) == CARRYF) state_err(outr.ax);
0424:     }
0425:     fpe("\n");
```

. . . listing continued

TRY MAGIC PC: THE ULTIMATE DBMS POWER

Attn Btrieve™ programmers:

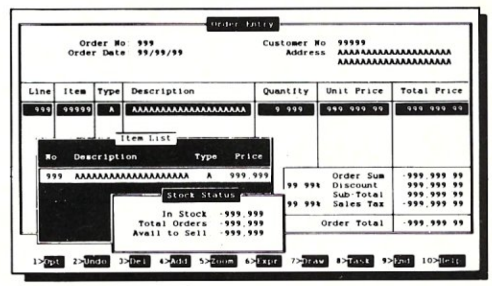
You know how database applications are created — by hacking out line after line of time-consuming code. Most DBMS' and 4GL's give you some programming power. But when it comes to serious applications, they keep you bolted to your seat writing mountains of tedious code. And rewriting it all over again with every design change.

Imagine how much faster you'd be if you could replace the painful coding phase with an innovative visual technology which takes only a fraction of the time: Introducing Magic PC—the revolutionary Visual Database Language from Aker Corporation:

TECH JOURNAL

"Magic PC's data base engine delivers powerful applications in a fraction of the time. . . there is truly no competitive product."

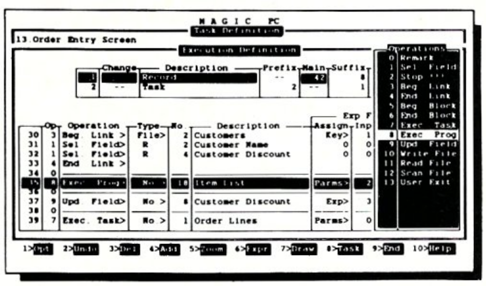
Victor Wright — PC Tech Journal



Pop-up Zoom Windows run multiple programs per screen — with point-and-shoot data transfer between windows!

• High-Speed Programming:

With Magic PC's visual design language you quickly describe your programs in non-procedural Execution Tables. They contain compact programming operations which are executed by Magic PC's runtime engine. You fill-in the tables using a visual interface driven by windows and point-and-shoot menus. One table with 50 operations eliminates writing more than 500 traditional lines of code. Yet with Magic PC you don't sacrifice any power or flexibility.



With a powerful set of high-level non-procedural operations you program at only a fraction of the time.

• Maximum Power AND Simplicity:

With Magic PC, you can generate robust DBMS applications including screens, windows, menus, reports, forms, import/export, and much more! Plus, Magic PC has one of the friendliest user interfaces you've ever seen. Using Magic PC you can look-up and transfer data through a powerful Zoom Window system. Magic PC even lets you perform command-free queries.

• Btrieve Performance:

Magic PC incorporates Btrieve, the high-performance file manager from SoftCraft. This gives you exceptional access speed, extended data dictionary capabilities, and automatic file recovery!

• Virtually Maintenance-Free:

With Magic PC you can modify your application design "on the fly" without any manual maintenance. Magic PC automatically updates your programs and data files on-line! This also makes Magic PC an ideal tool for prototyping complete applications in hours instead of days.

• FREE Networking:

Magic PC comes complete with LAN features. Develop multi-user applications for your LAN with Magic's file and record-locking security levels.

• Stand-Along Runtime:

Distribute your applications and protect your design with Magic PC's low cost runtime engine.

• All For Only $199:

Best of all, Magic PC is an unbeatable bargain. For a limited time, Magic PC's price has been reduced to only $199! Yes, this is the same Magic PC that normally lists for $695! And Magic PC eliminates the need for a separate DBMS, compiler, or application generator. It comes complete with all the tools you need to develop your own database applications instantly.

• $199 — With A Money-Back Guarantee!

For a limited time, you can get Magic PC for only $199. And even at this low price, Magic PC is risk-free. If you're not completely satisfied, simply return it within 30 days and we'll buy it back (less $19.95 restocking fee). And if you'd like a preview, Magic PC's Tutorial Demo is available for just $19.95.

But you'd better hurry — Magic PC's special $199 price won't last long!

• Join The Magic PC Revolution

To unleash your DBMS design power, order your $199 copy of Magic PC right now by calling toll-free or returning the coupon below.

ORDER NOW: CALL
(800) 345-MAGIC
In CA (714) 250-1718

MAGIC PC
The Visual Database Language
by AKER

Yes! I want to generate powerful applications much faster!

☐ Rush me my copy of Magic PC at the special promotional price of $199 (add $10 P&H, and tax in CA. International orders add $30). I understand I can return Magic PC for a refund within 30 days, if I'm not completely satisfied.*

☐ Rush me a copy of Magic PC Tutorial Demo at $19.95 (add $5 P&H, and tax in CA. International orders add $15).

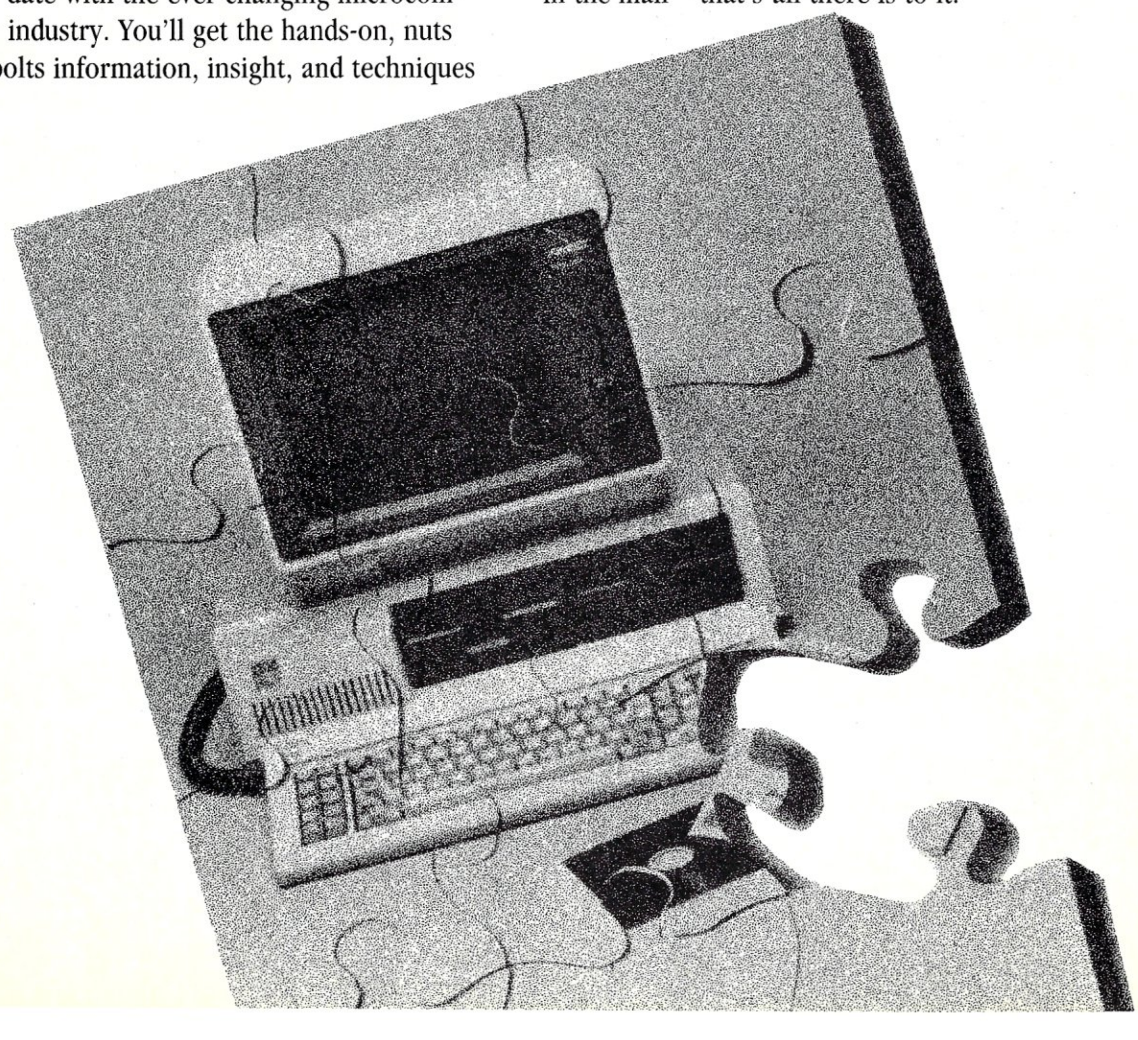
Name ______
Company ______ Phone ______
Street Address (no POB) ______
City ______ State ______ Zip ______
☐ Check enclosed Charge to my: ☐ VISA ☐ MasterCard ☐ AMERICAN EXPRESS
Account No. ______
Acct. Name ______ Exp. Date ___/___
Signature ______
Return to: **Aker Corp.**, 18007 Skypark Cir B2, Irvine, CA 92714

SMS

System requirements: IBM PC, XT, AT, PS/2 or 100% compatible with 512K RAM, hard disk and DOS 2.0 or later. 5¼" format, not copy protected. Dealer pricing available. *Return policy valid in US only.

```
}

/*----------------------- STATE_ERR ----------------------------*/

/* States error returned in AX register after change attribute call.
   Will abort if no file or incorrect path, since this indicates
   program can not start. Will not abort on "Access Denied", since
   this can happen with subdirectories.
*/

void state_err(rax)

unsigned rax;

{
void fpe();

    rax = rax & 0x00ff;                /* isolate AL register  */

    if (rax == NOFILE)   fpe("\nFile not found\007");
    if (rax == NOPATH)   fpe("\nPath not found\007");
    if (rax == NOACCESS) fpe("    Access denied\007");

    if ( !( (rax == NOFILE) || (rax == NOPATH) || (rax == NOACCESS) ) )
        fpe("\n\007Undefined error");

    if (rax != NOACCESS) {
        fpe("\n");
        exit(1);
    }
}


/*-------------------------- BAD_SYNTAX ------------------------*/

/* Indicates command line error and shows correct syntax
*/

void bad_syntax()

{
void fpe();

    fpe("\nIncorrect syntax\007");      /* 361 is octal for +/- */
    fpe("\n\nUsage: chmod [\361R | \361S | \361H] file file...\007");
    fpe("\n\n\361R = R/O on/off");
    fpe("\n\n\361S = SYSTEM on/off");
    fpe("\n\n\361H = HIDDEN on/off\n");
    fpe("\nN = change without asking");
    fpe("\nControls may be strung together (N must not come first)");
    fpe("\n   i.e. +R-HN+S or -S+RN");
    fpe("\n\nIf no control, then file attributes are displayed only\n");
    exit(1);
}

/*------------------------ DISPLAY_TYPE ----------------------*/

/* Displays attribute of file. Called from pfname() which gets
   attribute from dmabuf.attribute.
*/

void display_type(att)

char att;

{
void fpe(), cursor_col();

    cursor_col(30);

    if ( (att & 0x1f) == 0) fpe("NORMAL ");
    else {
        if ( (att & RO)     == RO) fpe("READ/ONLY ");
        if ( (att & HIDDEN) == HIDDEN) fpe("HIDDEN ");
        if ( (att & SYSTEM) == SYSTEM) fpe("SYSTEM ");
        if ( (att & VOLUME) == VOLUME) fpe("VOLUME ");
        if ( (att & SUBDIR) == SUBDIR) fpe("SUB DIRECTORY ");
    }
}

/*-------------------- FPE ----------------------------------*/

/* Prints string using MS-DOS console I/O function. This is
   because fputs() library function is incredibly slow. Also
   tests for console input and pauses if any key is pressed. Will
   resume when any key is pressed again.
*/

void fpe(string)

char    *string;

{
static char ch, chl;

    while ( (ch = *string++) != (char) NULL)
        if (ch != '\n') echo(ch);
        else {
            echo('\015');               /* ASCII CR */
            echo(ch);                   /* now do the LF */
        }

        if ( (chl = input()) != (char) NULL)
            if (chl == 3) exit(1);
            else while ( (chl = input()) == (char) NULL)
                if (chl == 3) exit(1);
}

/*------------------ CURSOR_COL ----------------------------*/

/* Places cursor at column "col" of current line using BIOS
   int 10h */

void cursor_col(col)

int col;

{
struct REGS inreg, outreg;
static unsigned current_page;

    inreg.ax = CURR_VIDEO;  /* gets active page in BH register */
    sysint(VIDEO_INT, &inreg, &outreg);
    current_page = outreg.bx;

    inreg.ax = READ_CURSOR; /* read cursor position to get row */
    inreg.bx = current_page;
    sysint(VIDEO_INT, &inreg, &outreg);

    inreg.dx = (outreg.dx & 0xff00) + (col & 0x7f); /* leave row */
    inreg.bx = current_page;                          /* alone and */
    inreg.ax = SET_CURSOR;                  /* .put cursor in col */
    sysint(VIDEO_INT, &inreg, &outreg);
}


/*------------------------- INPUT ----------------------------*/

/* Will return NULL if not character ready, or character. Does not
   wait for input. Uses MS-DOS function AH = 6 for raw console I/O.
   Can not be used for function key input.
*/

char input()

{
struct REGS inreg, outreg;
static int zerof;

        inreg.ax = CONIO;
        inreg.dx = 0x00ff;          /* input subfunction */
        zerof = sysint(MSDOS, &inreg, &outreg);
        if ( (zerof & ZEROF) == ZEROF) return (char) NULL;
        else return (char) outreg.ax;
}

/*---------------- ECHO ----------------------------------*/

/* Echos character to console using MS-DOS function AH = 6
*/

void echo(ch)

char ch;

{
struct REGS inreg, outreg;

    inreg.ax = CONIO;
    inreg.dx = (unsigned) ch;
    sysint(MSDOS, &inreg, &outreg);
}
```

End of Listing 2



Turbo C 1.5 versus Quick C 1.0

by Stephen R. Davis

Why so much excitement among C users over two new C compilers? It's not as if we haven't had C compilers before. They have been abundant since the days of CP/M. Small-C compilers have even been in the public domain, available to all for the price of a magazine. What makes these two products so different from the rest of the pack is their integration of editor and compiler into a combined development environment. From this environment, the programmer can find errors, correct them, and test the resulting program at a rate unapproachable with conventional compilers.

Borland started the practice with Turbo Pascal back in the early days of the PC. Turbo Pascal was essentially a small, very fast compiler that could generate, if not outstanding, at least reasonable code, molded together with a WordStar-like program editor. Turbo Pascal had several serious limitations: it understood only one memory model; it couldn't generate programs larger than 64K; and it couldn't be linked with other languages. Still, Turbo Pascal was a wild success. Programmers fell in love with the ease of use inherent in such a combined programming environment.

While Turbo Pascal was incrementally improved over the years, it was quite some time before the next integrated language appeared. Turbo C didn't appear on store shelves until the summer of 1987, followed only a few months later by Microsoft's version of the same thing, Quick C. How are these two "newest generation" compilers similar? What differentiates them from each other?

Table 1 compares the features of the two compilers. We will start by examining Turbo C 1.5. Then, due to the large number of similarities between this compiler and Quick C, we will point out only the important differences in Quick C. Finally, I will draw some admittedly subjective conclusions.

Turbo C 1.5

Turbo C is somewhat intimidating when you first open the package, especially if you are expecting it to be at all like Turbo Pascal. Tucked away within the shrink-wrapped manuals are five floppy disks. Although one contains nothing but examples, the combined package still consists of 834K of executable files, plus 100K of include files and 700K of object libraries. Since Turbo C itself makes up only 240K of that, and since not all of the object libraries are necessary at any one time, operation from 360K floppy disks is still just possible. Once you get beyond the size, actual installation of the package runs quite smoothly. A special install program allows the user to specify screen colors, adapter type, and editor commands. In addition to normal 25-line mode, Turbo C supports 43-line mode on EGA and 50-line mode on VGA adapters. (I found these modes too ragged to be easily readable.)

Especially useful is the ability to specify directories for each of the different support file types. For example, suppose Turbo C itself were located in C:\TURBOC. Rather than lump the whole package together, the user might prefer to place the include files in C:\TURBOC\INCLUDE and the .LIB library files in C:\TURBOC\LIB. The user simply supplies these directory names to Turbo C, and the compiler does the rest. If desired, these directories can be divided further, with the small model libraries in LIB\SMALL, the medium model libraries in LIB\MEDIUM, and so on. Turbo C allows a list of directories to be specified, and each directory is searched in turn until the desired file is found.

Turbo C allows the user to specify a working directory, where target and temporary files are stored. This capability allows the user to direct Turbo C to use a RAM disk, if available, as the temporary storage device—an advantage that leads to impressive increases in compilation speed. This capability also keeps temporary files from cluttering up the user's source directory.

Just as with Turbo Pascal, the Turbo C editor can be reinstalled for any commands desired. The default is for the WordStar command set plus another, more mnemonic, set of synonyms. For example, depressing the right arrow key might be somewhat more obvious than ^D. No macro capability exists for defining editor commands that do not already exist.

Once Turbo C is installed, it's ready for use. The Turbo C environment is very menu-oriented. Single key "short cuts" exist for many of the command sequences, but the drop-down menus are so fast and obvious, there seems little reason to memorize them. Context-sensitive help is available at almost every turn. This help is not of the all-encompassing, "replace the manual" variety, however.

By selecting the Open selection of the File menu, the user may pick any of the C files in the current directory, either by pointing or by entering its full name. The File menu also allows the user some simple DOS commands, such as changing subdirectories.

Interestingly, Turbo C maintains what is known as a Pick file. Every time a file is edited, Turbo C remembers the tab settings, any marked blocks, and the last position within the file. As soon as the user brings up Turbo C, it automatically loads the last file worked on and places the user at the last edit. The effect is to further enhance programmer speed.

As noted, the editor itself is WordStar-like in both its command set and its style of operation. Anyone familiar with the Turbo Pascal editor will feel right at home here. There are three additions worthy of note, however. First, the curious operation of the Tab key is now optional; under Turbo C, the Tab key can be made to tab over to user-selectable tab positions. Second, a new "pair matching" command has been added to help sort out parentheses, braces, and comments; the user places the cursor on a {, for example, depresses "pair match," and the editor instantly jumps to the corresponding }. Finally, a "go to previous/next error" command has been added and is quite convenient for moving from one compilation error another in order to quickly sort out compiler problems.

The compiler is also accessed from a pull-down window. The Turbo C compiler is completely compatible with the Kernighan and Ritchie implementation of C. In addition, Turbo C follows most of the ANSI draft standard, including function prototyping, enumerated data types, the VOID data type, the VOLATILE and CONST specifiers, and stronger typing.

Both in its library and in its language extensions, Turbo C tries very diligently to be upward-compatible with Microsoft C Version 4.0. Turbo C sup-

Table 1—Comparison of Quick C 1.0 and Turbo C 1.5 features

	Quick C	Turbo C
Interface		
interface		
interactive via keyboard	Y	Y
interactive via mouse	Y	N
command line	Y	Y
43-line EGA/50-line VGA	Y	Y
user-installable colors	Y	Y
built-in debugger	Y	N
help		
context-sensitive	N	Y
all-inclusive	Y	N
available from command line	Y	Y
pick files supported	N	Y
Language		
full Kerningham and Ritchie	Y	Y
ANSI proposed standard	Y	Y
Pascal fn. passing rules supported	Y	Y
case-insensitivity at link	Y*	Y
pseudo-variable access of registers	N	Y
assembly language output	N	Y*
inline assembly language	N	Y*
direct video text output w/*printf*	N	Y
memory models		
interactive	1+	6++
command line	4+++	6
mixed memory modes allowed	Y	Y
declarations		
interrupt procedure	Y**	Y
volatile const variables	Y**	Y
register variables	N	Y
double	Y	Y
long	Y	Y
Editor		
support WordStar commands	Y	Y
support mnemonic commands	Y	Y
commands redefinable	N	Y
Undo available	limited	limited
next/previous error	Y	Y
go to specific error	N	Y
tab size selectable	Y	Y
autoindent	Y	Y
autosave	N	Y
print command	Y	Y
pair matching	Y	Y

ports six memory models: small, medium, compact, and large, plus tiny and huge (tiny being similar to small with combined data and code segments, and huge being a special large memory model that allows arrays larger than 64K although at some cost to speed). Turbo C supports mixed memory model operation via NEAR and FAR pointer declarations of pointers and functions, independent of the default memory model. An additional function type, INTERRUPT, allows programmers to write interrupt and terminate-and-stay-resident (TSR) programs easily in Turbo C.

Turbo C can optionally generate 80186/286 real-mode instructions for machines that can handle it, and can utilize the 8087/287/387 floating point processor, if available. Even when present, Turbo C-generated programs can be instructed to ignore the floating point processor. This feature is highly desirable when testing floating point software.

Although it isn't obvious with simple programs, Turbo C's Compile command is actually a complete "make." When the user selects Compile, Turbo C compares the creation time of the object file with the time of the last source file edit. If the edit is more recent, Turbo C updates the object by compiling it. More complex dependencies, such as in multiple module pro-

	Quick C	Turbo C
Compiler		
operation		
number of passes	1	1
compilation speed [lines per min.]	10,000	>7,000
generates stand-alone .EXE	Y	Y
links with other languages	Y	Y
8087 support		
run-time sense	Y	Y
deselectable	N	Y
80186/286 instructions	Y	Y
warning levels		
user-selectable	Y	Y
independently controllable	N	Y
degree of compiler control	some	great deal
output-controllable	Y	N++++
debugger support		
Codeview	Y	N
other source code debuggers	Y	Y
SIEVE benchmark time ***	4.7	3.9
Miscellaneous		
MAKE from command line and environment	Y	Y
LINK		
from command line and environment	Y	Y
support overlays	Y	N
object .LIB librarian	Y	Y
graphics support library	Y	Y
library source code available	Y ($150)	Y ($150)
memory recommended	448K	384K
floppy operation possible	Y	Y
price	$99.00	$99.95

+ = Medium memory model only
++ = Tiny, small, compact, medium, large, and huge memory models
+++ = Small, compact, medium, and large memory models
++++ = Turbo C always sends output directly to disk; this may be redirected to a RAM disk, if available
* = command line version only
** = not documented
*** = time in seconds for 10 iterations of Byte Sieve on a 7.6 MHz IBM PC with all optimizations enabled under medium model

grams, are specified in a separate project file. For example, the user may specify that a module must be recompiled and the program relinked if an include file of a different name is edited. The effect is very efficient.

The same process—comparing the creation time of the object to that of the executable—is repeated in the link step. If any part of the object is newer, the program is relinked. By going through an explicit link step, Turbo C modules can be combined with object files of other languages. To facilitate this, Turbo C supports two different procedure-calling protocols: one for "C-type" procedures and another for "Pascal-type" procedures. In addition, Turbo C can be instructed to perform a case-insensitive link, treating all letters as if they were uppercase. (C is strictly case-sensitive, but most other languages are not.) The resulting executable file may be directed to any disk desired, even a RAM disk. Compiling or linking directly into RAM is not possible, however.

Turbo C allows a great deal of control of the compile/link process. The user may select which types of warning messages to print, what type of optimizations to perform, which memory model to use, what type of floating point, and so on. All of the features normally accessed via complex command-line switches are accessible from pull-down menus. Once made, these user preferences may be saved to disk.

Having created an executable file, the user may also choose to run the program directly from within the environment. Anyone who has ever lost a session of source file edits due to an errant program crashing the system will appreciate Turbo C's autosave feature, which automatically saves edited files to disk before the user program is executed.

Executing programs from within the Turbo C environment is a bit of a problem, however, since Turbo C has no built-in debugger. This omission is difficult to understand, especially since the absence of a debugger was sorely felt under Turbo Pascal. It seems that Borland would have been quick to add a debugger in Turbo C. Fortunately, Turbo C will generate a detailed load map. This allows Turbo C-generated programs to be used with third-party source-code debuggers, such as Periscope or PFix. Turbo C does not directly support the Codeview debugger, however.

I found that I could shell to DOS from the File menu and then execute my program under PFix to debug it. Once errors were found, I simply exited PFix, entered EXIT at the DOS prompt and, pop, I was back in Turbo C. Although this is quick enough, it should not be necessary. Borland is rumored to have a debugger in the wings and we can only hope that it is available by the time this article appears.

For those that require it, a separate command-line version of Turbo C is available. This version includes all the language features of the environment version, plus two others: (1) the ability to generate assembly language output and (2) the ability to include inline assembly language. These are powerful features that come in quite handy, especially when debugging Turbo C INTERRUPT procedures. Complementing this command-line version of the compiler are command-line MAKE and TLINK facilities. One extra utility, not available from within the environment, allows users to create and maintain their own object libraries. Unfortunately, preferences established and recorded from the environment version do not automatically carry over to the command line. Instead, users must build involved batch files specifying chains of switches. Invoking it without any arguments prompts the compiler to print a summary of all the legal switches, making some trips to the reference manual unnecessary.

> *Turbo C modules can be combined with object files of other languages.*

Like previous Borland manuals, the three Turbo C manuals are bound softbacks in the slightly larger, European format. While all the information appears to be there, Turbo C's manuals are slightly scatterbrained, sometimes leaving a subject only to return to it several chapters later.

Unfortunately, Borland did not revise the Version 1.0 manuals for Version 1.5, choosing instead to add a third manual, *Enhancements*, that describes all the improvements and changes in Version 1.5 over its older sibling. References are made in the *Enhancements* manual to specific page numbers in the other two books. Unfortunately, some things did change in the other manuals, so page references are not accurate. Normally, this would be unacceptable in manuals, but since most changes between 1.0 and 1.5 are actually additions, the situation is not too bad.

The Turbo C library is quite extensive. Borland makes the source code available at extra charge. This can be useful in a commercial application, where the user might want to modify the operation of certain library routines. Included in the library is a graphics manager that supports all of the common display adapters including VGA, EGA, Hercules, and AT&T.

Quick C 1.0

In many respects, Quick C is similar to its Turbo C rival. As imposing as Turbo C was when opening the box, Quick C is even more so. Quick C arrives encased in two manuals: a bound, all-inclusive language description and a compact, three-ring binder description of the library routines.

Installation of Quick C onto a hard disk or user floppies is performed via an install program. Before it can be installed, however, users must decide several issues with which they may not be familiar. As I later found out, changing these decisions is not particularly difficult but must be performed manually. Once Quick C is physically resident in the machine, no further installation is necessary.

As with Turbo C, Quick C allows include and library files to be in different directories; however, Quick C executes this by setting labels in the environment. This means that, before running Quick C, a batch file must be used to set these labels up and these decisions cannot be changed once in the environment. In addition, only single directories are allowed.

Like Turbo C, Quick C is menu-oriented. Unlike its competitor, Quick C supports a mouse interface in a very Windows-like fashion. Although operation without a mouse is possible, the feel is somewhat "klunky." With a mouse, the package really shines. I am still no fan of editing with rodents, but manipulation of the Quick C environment is a dream. Quick C also allows single-key combination access to many of the commands. Help, available most of the time, is not context-sensitive. Instead, it is all-inclusive, with its own menu of offerings that even describes in some detail individual library routines.

The Quick C editor uses the WordStar command set in addition to a more mnemonic set of commands. In addition, many editing functions can be performed with the mouse. The Quick C editor includes all the features of the Turbo C editor including autoindent, "pair matching," block move, and print. The editing commands are not user-installable. Quick C's tab is the normal variety and the user can select the spacing. Quick C's autoindent feature cannot be disabled. Like Turbo C, Quick C's Undo command is limited; however, since most deletes go to the ClipBoard, inadvertent deletes can be pasted back in place without undue heartache.

The Quick C compiler is compatible with Microsoft C 5.0. This means that it implements the Kernigham-and-Ritchie standard and the ANSI draft standard extensions. Quick C supports the four important memory models but supports only the medium memory model from within the environment. As with Turbo C, Quick C can generate 80186/286 real-mode instructions and will support a 8087/287/387 floating point processor. It is not possible to deselect the floating point processor, if present.

The Quick C compiler supports a built-in make that is much like that in the Turbo C compiler. Program lists with enumerated dependencies are entered in a similar fashion. Although the user can control the level of warning messages and some optimization features, the level of control from within the environment is not as high as it is with Turbo C. User preferences may be saved off, but they are always stored in a file called QC.INI in the current directory.

Quick C may compile to disk, to memory, or, in what is known as "syntax check only" mode, to the bit bucket. Compilations to memory are very fast. The excellent debugger built into the Quick C environment is a subset of the Codeview debugger. Commands are available to single step, set breakpoints, and set watch variables. The Quick C debugger, especially when used with a mouse, simplifies debugging under the environment many fold. It is elegant in feel and operation. For those more tenacious bugs, Quick C supports Codeview (although it is not included). As with Turbo C, a separate command-line version of Quick C is available. That supports the memory models unsupported by the environment version. Otherwise, the compilers seem identical.

Also available are command-line versions of MAKE, LINK, and a library utility. The Microsoft linker supports automatic overlay creation and maintenance, an important feature with really large programs.

The Quick C manuals are very well written. They are just as notable, however, for what they leave out or men-

The West Coast Computer Faire announces the first Computer Matchmaking Service.

You won't have to depend on fate at the 13th West Coast Computer Faire to find the products and services that are the perfect match for your needs.

We start you out on your path to high-tech bliss with Vertical Market Matching. We bring in the companies selling quality computers, software, peripherals, and add-ons—companies that meet the needs of people involved in specific business segments such as finance, medicine, manufacturing, law, education, engineering, and construction.

And our Product Matching makes it easy for you to find the software, add-ons and upgrades for the Commodore Amiga, Apple II or Macintosh, IBM PC/MS-DOS, IBM PS/2, Atari, Lotus and more, that will keep you happily gazing into your current system's eyes. Plus, we counsel you on the latest techniques and insights in our outstanding Conference sessions.

The West Coast Computer Faire has made and will make more matches than any another computer show. It's time we made the perfect match for you.

Match your interests with these Faire Features:

- Computer Art Gallery
- Computer Faire Networks
- Computer Music Demo
- Computer-Aided Special Effects Demo
- Exhibitor Presentations
- Free Hands-On Classes, including Desktop Publishing, MS-DOS, Word Processing, Lotus 1-2-3

The West Coast Computer Faire, Moscone Center, San Francisco, CA, April 7-10, 1988

For information on exhibiting, call 617-449-6600, x5077. But hurry — the Faire's floor is almost full!

Register early and save $5!

Fill out this coupon and mail with your check(s), for $15.00 for each registrant, postmarked by March 17th, 1988. Include the names and addresses of registrants for whom you are enclosing a check. (Photocopy coupon for additional registrants.)

Name ______________________ Title ______________________

Company ____

Address ____

City ____________________ State _______ Zip ______________

Phone (__________)____________________________________

Four day conference and exhibits $15.00 in advance. $20.00 at the door. Make check payable to "The West Coast Computer Faire." Mail to: Attendee Registration Department, The West Coast Computer Faire, 300 First Avenue, Needham, MA 02194. Advanced registrations accepted only with full payment and each registrant's name and address. Tickets will be mailed to each individual registrant separately.

THE 13TH WEST COAST COMPUTER FAIRE

April 7–10, 1988, Moscone Center, San Francisco, CA

tion only briefly as for what they contain. For example, the fact that the interactive version of Quick C supports only the medium memory model is buried about as deeply as humanly possible. Several features supported in Quick C (for example, INTERRUPT procedures) are not mentioned at all.

All this leads to the fact that Microsoft has a problem. Microsoft claims that Quick C is "100% Microsoft C 5.0 compatible," which means it must support 5.0 features whether the manual mentions it or not. In fact, the object libraries for Quick C are exactly the same as the ones provided with C 5.0. What, then, is the incentive to purchase the 5.0 package for roughly four times the price? Unfortunately, not documenting certain features in order to make them appear absent, as well as arbitrarily limiting the interactive version to one memory model, might provide such a buying incentive—even when the software itself does not.

Conclusions

Performance of the two compilers is similar. Both compile quickly, generating reasonably efficient code. The time difference in executing the sieve of Eratosthenes is due primarily to Turbo C's register optimization—Quick C showed little or no register optimization and ignored the REGISTER directive when present.

I could not come up with a clear pick. Had Turbo C included a built-in debugger, its ease of use and wealth of user control would have been made it my hands-down winner, even without mouse support. If Quick C had included more control of directories and compiler features from within the environment, and had Microsoft not purposely shackled the package, Quick C with its mouse interface and its fine debugger would have come out on top.

I might also point out that the availability of Microsoft C 5.0 does not change my opinion much. Despite Microsoft's attempts, I cannot find much to differentiate C 5.0 from Quick C—other than two extra manuals and the included Codeview debugger. If you already have Codeview, stay with Quick C and borrow the extra manuals when you need them. In addition, I would still put Turbo C plus any good source code debugger up against Microsoft C 5.0, despite the disparity in price.

There being no overwhelming winner, I applaud both packages. While neither is without flaws, both compilers are excellent, first-class development tools at a very reasonable price. C programmers should be ecstatic. □

Stephen Randy Davis is technical editor for Micro/Systems, *and a programmer for a defense contractor in Greenville, Texas.*

Product Information

Microsoft Corp
16011 N.E. 36th Way
Box 97017
Redmond, WA 98073
(206) 882-8080

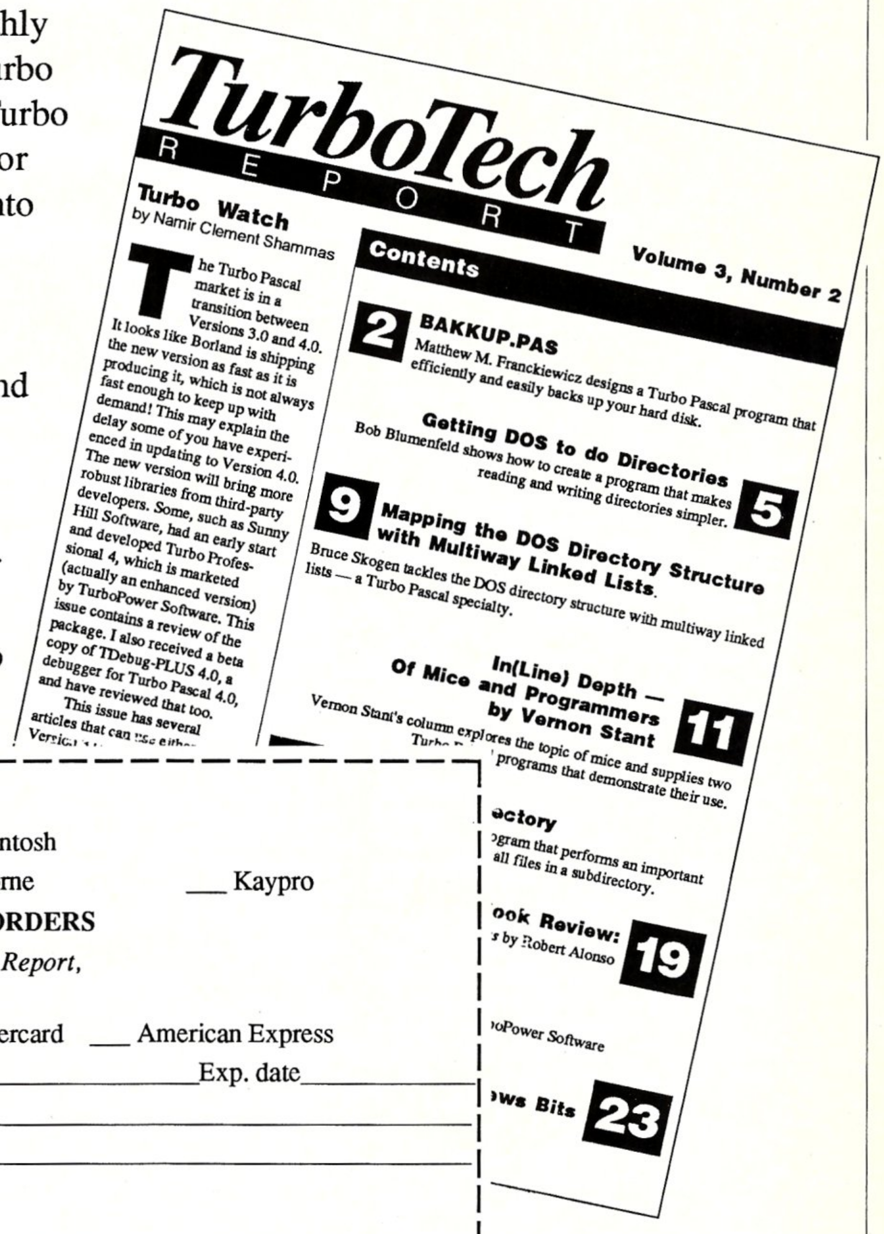
Borland International
4585 Scotts Valley Dr.
Scotts Valley, CA 95066
(408) 438-8400

Expand your skills with Turbo Tech Report

Turbo Tech Report is a bimonthly newsletter/disk publication for Turbo Pascal programmers. If you use Turbo Pascal regularly and are looking for powerful utilities to incorporate into your own programs, *Turbo Tech Report* is for you.

Each bimonthly issue delivers useful programming techniques and tools on paper <u>and</u> on disk — articles, reviews, Product-in-Action tidbits, and source code on disk.

Subscribe now and receive our special spring discount — just $89.00 for a year's worth of Turbo Pascal tools!



I need the following disk format (check one):

___ PC-DOS/MS-DOS ___ Macintosh

CP/M: ___ Apple ___ Osborne ___ Kaypro

PAYMENT MUST ACCOMPANY ALL ORDERS

____ Check enclosed, payable to *Turbo Tech Report*, for $89.00. ($129 for orders outside U.S.)

Please charge my: ___ Visa ___ Mastercard ___ American Express

Card #____________________ Exp. date________

Name on Card____________________

Signature____________________

Please mail the subscription to:

Name____________________

Address____________________

City____________ State______ Zip______

Please allow four to six weeks for subscription to begin.

A publication of M&T Publishing, Inc. 3APR

Mail this coupon to M&T Publishing, 501 Galveston Drive, Redwood City, CA 94063

The Atron MiniProbe I Debugger

An inexpensive, hybrid debugging solution.

by Michael Guttman and Bruce Gould

Nearly every serious programmer owns some sort of debugger. Although software debuggers are not usually expensive, even a modestly capable hardware debugger can cost $1,000. Since this may often be more expensive than the computer itself, one might think twice before buying.

Atron now offers the MiniProbe I, a hybrid debugging solution that combines Atron's sophisticated symbolic debugging software with enough hardware to give the feel of a "real" hardware debugger, all for $395. In addition, the MiniProbe offers a direct hook into Microsoft's Codeview, one of the most sophisticated and popular source-level debuggers around—and an indirect interface to Microsoft's Symdeb debugger.

We tested the MiniProbe I over a period of several months, using it in a wide variety of situations (including debugging a number of assembly language TSR programs we have under development). We also tested the MiniProbe with Codeview.

Installation

We installed the MiniProbe I on an IBM AT, although the MiniProbe also will work with any PC or XT. The Atron hardware consists of a half-size plug-in card and a crash recovery switch box. Physical installation involves inserting the board into any available expansion slot and then connecting the switch box via a connector at the rear of the board. A jumper setting on the board allows the user to change the MiniProbe's I/O port settings if necessary to avoid conflicts with other boards.

Although installation proceeded smoothly enough, we noticed that the six-pin switch box connector on the board is very flimsy and easy to connect incorrectly. Several times, normal adjustments to the machine—taking off the cover for servicing or adding a new board—unseated the connector. This did not happen to us because we were careful not to place any stress on the connection.

The MiniProbe hardware also allows the option (for ATs only) of reconnecting the computer's reset switch to the MiniProbe switch box. This allows the user to reboot the computer without having to power down when the normal Ctrl-Alt-Del key sequence won't work. However, since the instructions were unclear and warned about damaging our computer, we decided not to install this option.

Once the MiniProbe hardware is in place, the user can test the MiniProbe by running the CHECKOUT.EXE program provided on the software diskette. The diskette also contains the DEBUG.SYS device driver used to interface with Codeview and the WATCH.EXE program used to interface with Symdeb. (Use of the Codeview device driver naturally requires a DEVICE=DEBUG.SYS entry in your CONFIG.SYS file.)

While installation of the MiniProbe on our IBM AT was successful, we could not install the MiniProbe on our Televideo AT. In short, the MiniProbe simply does not work with the Televideo, and, according to Atron, it also won't work with a number of other clones, including the Zenith 241 and 248, the Sperry IT, the Compaq Portable 286, the ITT Xtra, the Panasonic AT, or the PC's Limited AT. Clone owners should check with Atron to find out if their machine has been tested before buying the MiniProbe.

Operating the MiniProbe with Codeview and Symdeb

There are three software options that can be used with the MiniProbe: Codeview, Symdeb, or Atron's Software Source Probe. Hardware settings are the same for the three modes, so intermingling the three options on the same machine to solve different problems is quite practical.

The MiniProbe interacts with Codeview via the DEBUG.SYS device driver. When Codeview is active, the MiniProbe hardware will automatically take over watching one memory area as defined in a Codeview watchpoint or tracepoint setting.

The main advantage of the MiniProbe in this configuration is that it can perform a checking operation up to 1,000 times faster in hardware (according to Atron sources) than the Codeview software alone. This extra speed is particularly useful if the variable being checked is nested deep down and called by many routines. Debugging that could otherwise take minutes or hours can be reduced to seconds.

A basic limitation of the MiniProbe I is that it can only watch one memory area at a time, making it less useful if you need to watch for multiple memory areas. However, the MiniProbe interface will always check the last variable mentioned in a watchpoint

Quit Wasting Time!

As a programmer, most of your time is spent writing and debugging source code, and documenting your work. A powerful, easy-to-use programmable text editor could be saving you HOURS of unnecessary effort.

Only MULTI-EDIT has all these time-saving features:

Fully automatic Windowing and Virtual Memory.
Edit multiple files regardless of physical memory size.
Easy cut-and-paste between files.
View different parts of the same file.

Powerful, EASY-TO-READ high-level macro language.
Standard language syntax.
Full access to ALL Editor functions.
Automate repetitive tasks.
Easy, automatic recording of keystrokes.

Language-specific macros for ALL major languages.
Smart indenting.
Smart brace/parenthesis/block checking.
Template editing.
Supports C, Pascal, BASIC and Assembler.

Terrific word-processing features for all your documentation needs.
Intelligent word-wrap.
Automatic pagination.
Full print formatting with justification, bold type, underlining and centering.
Even a table of contents generator.

Compile within the editor.
Automatically positions cursor at errors.
Built-in MAKE capabilities.
Run compiled program without leaving editor.
Automatically allocates all available memory to compiler or program.

Complete DOS Shell.
Scrollable directory listing.
Copy, Delete and Load multiple files with one command.
Background file printing.

Regular expression search and translate.
Condensed Mode display, for easy viewing of your program structure.
Pop-up FULL-FUNCTION Programmer's Calculator and ASCII chart.

and MOST IMPORTANT, the BEST user-interface on the market!

- Extensive context-sensitive help.
- Choice of full menu system or logical function key layout.
- Function keys are always labeled on screen (no guessing required!).
- Excellent online, interactive tutorial.
- Keyboard may be easily reconfigured and re-labeled.

Users of Wordstar and Turbo Pascal's Editor could be programming in a fraction of the time with these features.

NO EDITOR ON THE MARKET TODAY HAS ALL THESE FEATURES, OR OFFERS YOU THIS MUCH POWER AT A REASONABLE PRICE, EXCEPT

Multi-Edit $99.
VERSION 2.0

COMPLETE
Or Get our FULLY FUNCTIONAL DEMO Copy for only $10!

	Multi-Edit	BRIEF 2.0	Norton Editor	Vedit Plus
Edit 20+ files larger than memory	Yes	Yes	No	Yes
Powerful high level macro language	Yes	Yes	No	Yes
Full UNDO	Yes	Yes	No	No
Visual marking of blocks	Yes	Yes	Yes	No
Column oriented block operations	Yes	Yes	No	No
Automatic file save	Yes	Yes	No	No
Online help	Extensive	Limited	Limited	Limited
Online tutorial	Yes	No	No	Yes
Choice of keystroke commands or menu system	Yes	No	No	Yes
Function Key assignments labeled on screen	Yes	No	No	No
WP Functions	Extensive	Limited	Limited	Extra Cost
Complete DOS shell	Yes	No	No	No
Pop-up Programmer's Calculator and ASCII table	Yes	No	No	No ASCII
Unlimited 'Off the Cuff' keystroke macros	Yes	No	No	Yes
Allocates all available memory to compiler when run from within editor	Yes	No	Yes	Yes
Intelligent indenting, template editing and brace/parenthesis/block matching and checking for all major languages	Yes	C Only	No	Limited
Flexible condensed mode display	Yes	No	Yes	No
Optional background communications and Spell Checker modules	Yes	No	No	No
PRICE	$99	$195	$50	$185

Requires IBM/PC/XT/AT/PS2 or full compatible, 256K RAM, PC/MS-DOS 2.0 or later-Multi-Edit and American Cybernetics are trademarks of American Cybernetics. BRIEF is a trademark of Underware, Inc. Norton Editor is a trademark of Peter Norton Computing, Inc. Vedit is a registered trademark of CompuView Products Inc. Copyright 1987 by American Cybernetics.

Get our FULLY FUNCTIONAL DEMO Copy for only $10!

To Order, Call 24 hours a day:
1-800-221-9280 Ext. 951
In Arizona: 1-602-890-1166
Credit Card and COD orders accepted

American Cybernetics
138 Madrid Plaza
Mesa, AZ 85201

statement, allowing a clever user to construct his statements with the most accessed variable last.

The procedure for using the MiniProbe with Symdeb, although less elegant, is not difficult. It consists of running the WATCH.EXE program from within Symdeb using the DOS command line option. Depending on the command-line options selected, the user can set a watchpoint for either memory or I/O addresses.

The Atron Software Source Probe

Although many programmers may be interested only in the MiniProbe's interface to Codeview, Atron's product also comes with a complete debugger called the Software Source Probe that debugs assembly programs and programs written for other compilers. Like Codeview or Symdeb, the Source Probe will run without the MiniProbe, but receives a useful assist from the hardware.

The Source Probe can be run in two modes: as a regular program and as a resident program. As a regular program, the Probe can be used to load and run an application so that when the STOP button on the switch box is pressed, it will freeze the application. If the Probe runs as a TSR (terminate-and-stay-resident program), the application to be debugged is started in the usual way, from the DOS prompt. When the STOP button is pressed, the Probe springs to life and freezes the application. Now the user can leisurely examine the state of the system, execute the application one instruction at a time, or let it resume running in the normal mode.

We found the Atron Probe run in the TSR mode invaluable in debugging our own TSR applications. In particular, we used the Probe's ability to set breakpoints in our code by calling interrupt 3. When the Probe was running as a TSR, an application calling interrupt 3 causes the Probe to pop up and the application to freeze. Unfortunately, this feature is not explicitly mentioned in the documentation—we discovered it intuitively based on information hidden in an appendix of technical information.

One of the TSRs we debugged took control of the keyboard and caused frequent freeze-ups of the system when we used the Probe. (Atron does warn of this possibility.) In order to avoid this problem, we used the remote-console feature of the debugger.

By connecting the COM port on our PC/AT via an RS-232 cable to another PC running a terminal emulator provided by Atron, the debugger screen appears and debugger commands are accepted from the second PC, leaving the first PC free to handle only the I/O of the user's application. This arrangement effectively allowed us to type in debugger commands on the second keyboard without interfering with the TSR program.

Unfortunately, the pin settings required on the connecting cable are nonstandard, and not properly documented. Only after contacting Atron

Table 1. Representative Source Probe Commands

BREAKPOINT—Defines a breakpoint at a given address or symbol. The program will run until a breakpoint is encountered.

BYTE—Displays the contents of memory at a certain address. For example,

```
by 0010:0010 l 10
```

will display 16 bytes of memory, in hex, starting at location 110h. (Addresses are expressed using the standard segment:offset method.)

POINTER—Displays the contents of memory as if the memory contains addresses in the segment:offset notation. (We found this useful in examining the values of interrupt vectors in low memory.) For example, the command

```
ptr 0000:0000
```

on our machine yielded the following line:

```
019a:4ee8 0f7a:0298 0f7a:01bb 0f7a:0260
```

from which we infer that interrupt zero points to 019a:4ee8.

R—Displays the values of all registers and flags.

for additional information and making a special cable did we get the remote-terminal feature to work.

Some debuggers, like Codeview, have screen windows that allow for easy monitoring of system information. Unfortunately, the Atron Source Probe has no such facility. For example, the R command must be typed each time the user needs to view the values of all registers and flags. As the debugger is used, the information on the screen scrolls up and disappears. The R command must be repeatedly typed to monitor the registers.

However, the Source Probe does have a macro language that can store a number of debugging commands as a unit and execute them by invoking the macro name. This language has simple procedural capabilities, e.g., there are IF and LOOP commands that test conditions and cause the conditional execution of macro commands.

The macro language somewhat mitigates the disadvantage of not having a window. Continually monitoring registers provides the ability to execute automatically via a macro after each GO or STEP command (see Table 1). This frees the user from constantly having to type R, for example, in order to see the registers after each instruction has been executed. However, it does not create a very pretty display.

The Source Probe also constantly displays a "help" area at the bottom of the screen. As soon as a typed command is recognized by the debugger, the syntax and the options of the command appear on the bottom of the screen.

Documentation and Support

The least satisfying component of the MiniProbe I and the Source Probe is the documentation. In addition to the examples cited above, we found the documentation lacks clarity and accuracy. As a result, we had to experiment with the program, scrutinize the manuals, or call technical support to uncover many of the most useful features of the software.

Fortunately, the quality of the technical help we got over the phone from Atron was very good. The staff took the time to walk us through problems and explain obscure points. Despite some very annoying documentation problems, we think the Atron MiniProbe I is a solid product and a good value. □

The authors are consultants with Morrisey Associates, a software development and consulting firm based in Skokie, Illinois.

Product Information

MiniProbe I $395

Atron Division
Northwest Instument Systems Inc.
20665 Fourth St.
Saratoga, CA 95070
(408) 741-5900

UNASSEMBLE—Displays the assembly code at a specified address. For example,

```
u 019a:4ee8
```

will display the code that interrupt zero points to.

STEP—Steps through code one instruction at a time. After this instruction is entered, pressing the Enter key will cause one instruction to be executed, until a non-Enter key is pressed. Variations of this command will allow the programmer to step over interrupts or called procedures. In its simplest form, the step command will trace through *all* code, including the operating system interrupt code.

GO—Executes the program until a certain address or breakpoint is reached.

PORT—Displays or modifies the contents of an I/O port.

MOVE—Moves a block of memory, specified by a starting and ending address, to a new location.

LIST—Copies the output of Probe commands to another device, giving a history of a debugging session.

Listing continued from page 27

```
                DBUGFLG6 = true;
            }
        } else {
            break;
        }
        C = fgetc(FLAGHNDL);
        if ( C != EOF) {
            if (C == '1') {
                DBUGFLG7 = true;
            }
        } else {
            break;
        }
        C = fgetc(FLAGHNDL);
        if ( C != EOF) {
            if (C == '1') {
                DBUGFLG8 = true;
            }
        } else {
            break;
        }
        break;
    }
}
#endif

        /*  Set debug level flags from environment */

#ifdef ENVFLAG
        FLAGPTR = envfind("DBUGFLG0");
        if (FLAGPTR) {
            strcpy(FLAGLIT,FLAGPTR);
            if (strcmp(upper(FLAGLIT),"TRUE") == 0) {
                DBUGFLG0 = true;
            }
            free(FLAGPTR);
        }
        FLAGPTR = envfind("DBUGFLG1");
        if (FLAGPTR) {
            strcpy(FLAGLIT,FLAGPTR);
            if (strcmp(upper(FLAGLIT),"TRUE") == 0) {
                DBUGFLG1 = true;
            }
            free(FLAGPTR);
        }
        FLAGPTR = envfind("DBUGFLG2");
        if (FLAGPTR) {
            strcpy(FLAGLIT,FLAGPTR);
            if (strcmp(upper(FLAGLIT),"TRUE") == 0) {
                DBUGFLG2 = true;
            }
            free(FLAGPTR);
        }
        FLAGPTR = envfind("DBUGFLG3");
        if (FLAGPTR) {
            strcpy(FLAGLIT,FLAGPTR);
            if (strcmp(upper(FLAGLIT),"TRUE") ==0) {
                DBUGFLG3 = true;
            }
            free(FLAGPTR);
        }
        FLAGPTR = envfind("DBUGFLG4");
        if (FLAGPTR) {
            strcpy(FLAGLIT,FLAGPTR);
            if (strcmp(upper(FLAGLIT),"TRUE") == 0) {
                DBUGFLG4 = true;
            }
            free(FLAGPTR);
        }
        FLAGPTR = envfind("DBUGFLG5");
        if (FLAGPTR) {
            strcpy(FLAGLIT,FLAGPTR);
            if (strcmp(upper(FLAGLIT),"TRUE") == 0) {
                DBUGFLG5 = true;
            }
            free(FLAGPTR);
        }
        FLAGPTR = envfind("DBUGFLG6");
        if (FLAGPTR) {
            strcpy(FLAGLIT,FLAGPTR);
            if (strcmp(upper(FLAGLIT),"TRUE") == 0) {
                DBUGFLG6 = true;
            }
            free(FLAGPTR);
        }
        FLAGPTR = envfind("DBUGFLG7");
        if (FLAGPTR) {
            strcpy(FLAGLIT,FLAGPTR);
            if (strcmp(upper(FLAGLIT),"TRUE") == 0) {
                DBUGFLG7 = true;
            }
            free(FLAGPTR);
        }
        FLAGPTR = envfind("DBUGFLG8");
        if (FLAGPTR) {
            strcpy(FLAGLIT,FLAGPTR);
            if (strcmp(upper(FLAGLIT),"TRUE") == 0) {
                DBUGFLG8 = true;
            }
            free(FLAGPTR);
        }
#endif

        /* Set file to nul unless at least one debug flag > 0
           is true */

        if ( ! (DBUGFLG1 || DBUGFLG2 || DBUGFLG3 || DBUGFLG4 ||
                DBUGFLG5 || DBUGFLG6 || DBUGFLG7 || DBUGFLG8)) {
            strcpy(DBUGNAME,"nul");
        }
        if (DBUGFLG0) {
            DBUGHNDL = stderr;
        } else {
            DBUGHNDL = fopen(DBUGNAME,"w");
            if (DBUGHNDL == 0) {
                fprintf(stderr,"\nDBUGOPEN: * Abort at 'fopen' - ");
                fprintf(stderr,"Error Return is %d *\n",
                    ferror(DBUGHNDL));
                exit(1);
            }
        }
        DBUGTIME();
        fprintf(DBUGHNDL,
            "DBUGOPEN: Debug Started with SCCSID='%s'\n",SCCSID);
        DBUGINT(FUNC,"DBUGFLG0",DBUGFLG0);
        DBUGINT(FUNC,"DBUGFLG1",DBUGFLG1);
        DBUGINT(FUNC,"DBUGFLG2",DBUGFLG2);
        DBUGINT(FUNC,"DBUGFLG3",DBUGFLG3);
        DBUGINT(FUNC,"DBUGFLG4",DBUGFLG4);
        DBUGINT(FUNC,"DBUGFLG5",DBUGFLG5);
        DBUGINT(FUNC,"DBUGFLG6",DBUGFLG6);
        DBUGINT(FUNC,"DBUGFLG7",DBUGFLG7);
        DBUGINT(FUNC,"DBUGFLG8",DBUGFLG8);
        fprintf(DBUGHNDL,"\n");
        fflush(DBUGHNDL);
    } else {
        DBUGTIME();
        fprintf(DBUGHNDL,"DBUGOPEN: Debug File Already Open\n");
        fflush(DBUGHNDL);
    }
} /* end DBUGOPEN */
/*----------------------------------------------------*/
public void DBUGPTR(P_FUNC,P_NAME,P_VALUE)

    string   P_FUNC[];          /*  in */
    string   P_NAME[];          /*  in */
    char    *P_VALUE;           /*  in */

    /*  Print pointer name P_NAME value P_VALUE in
        function P_FUNC. */
{
    static string               FUNC[] = "DBUGPTR";
    unsigned int                SEGMENT;
    unsigned int                OFFSET;
    unsigned long int           ABSOLUTE;

    extern unsigned long int    ptrtoabs();
/* begin */
    if (! DBUGOPFL) {
        DBUGOPEN();
    }
    DBUGTIME();
    fprintf(DBUGHNDL,"%-8s: %-16s=",P_FUNC,P_NAME);
#ifdef _C86_BIG
    SEGMENT  = (unsigned int)(((unsigned long int)P_VALUE) >> 16);
    OFFSET = (unsigned int)P_VALUE;
    ABSOLUTE  = ptrtoabs(P_VALUE);
    fprintf(DBUGHNDL," Ptr  Segm=%04x Ofst=%04x  ",
        SEGMENT,OFFSET);
    fprintf(DBUGHNDL,"Abs=%05lx\n",ABSOLUTE);
#else
    OFFSET = (unsigned int)P_VALUE;
    fprintf(DBUGHNDL," Ptr  Ofst=%04x\n",OFFSET);
#endif
    fflush(DBUGHNDL);
}   /*  end DBUGPTR */
/*----------------------------------------------------*/
public void DBUGSTAK(P_FUNC,P_NUM)

    string  P_FUNC[];           /*  in */
    int     P_NUM;              /*  in */

/* Print stack presented to function P_FUNC: base pointer
   (actually generated by P_FUNC), return address (big
   memory model), P_NUM words of parameter entries. */
```

```
{
    static string               FUNC[] = "DBUGSTAK";
    unsigned short int          BP;     /*  base pointer */
    unsigned short int          WP;     /*  work pointer */
    unsigned short int          SS;     /*  stack segment */
    unsigned short int          VAL;    /*  stack entry value */
    string                      CSTR[4+1]; /*  stack entry value */
    struct segregs              SEGS;   /*  segment registers */
    unsigned int                I;

/* begin */
    if (! DBUGOPFL) {
        DBUGOPEN();
    }
    BP = BASEPTR();
    segread(&SEGS);
    SS = SEGS.sss;
    BP = peek(BP,SS);       /* chain to stack presented to
                               function that called DBUGSTAK */
    DBUGTIME();
    fprintf(DBUGHNDL,"%-8s: %-16s  Address    Hex      Int  Char\n\n",
        P_FUNC,"Stack");
    for (I = 0 ; I < P_NUM + 3 ; I++) {
        WP = BP + (2 * I);
        VAL = peek(WP,SS);
        DBUGXLCH(CSTR,VAL);
        fprintf(DBUGHNDL,"%37s%04x:%04x  %04x%  %6d  %s\n",
            " ",SS,WP,VAL,VAL,CSTR);
    }
    fprintf(DBUGHNDL,"\n");
    fflush(DBUGHNDL);
}   /*  end DBUGSTAK */
/*------------------------------------------------------*/
public void DBUGSTR(P_FUNC,P_NAME,P_VALUE)

    string  P_FUNC[];           /*  in */
    string  P_NAME[];           /*  in */
    string  P_VALUE[];          /*  in */

    /* Print string name P_NAME value P_VALUE in
       function P_FUNC. */
{
    static string   FUNC[] = "DBUGSTR";
    int             I;
    string          TEMP[256 + 1];

/* begin */
    if (! DBUGOPFL) {
        DBUGOPEN();
    }
    DBUGTIME();
    fprintf(DBUGHNDL,"%-8s: %-16s=",P_FUNC,P_NAME);
    DBUGXLST(TEMP,P_VALUE);
    fprintf(DBUGHNDL,"'%s'\n",TEMP);
    fflush(DBUGHNDL);
}   /*  end DBUGSTR */
```

Listing 3. BASEPTR.ASM; Get base pointer

```
;  C language calling sequence for this function is:
;           unsigned int BP;
;
;           BP = BASEPTR();
;
            include model.h             ;C86
            include prologue.h          ;C86
            public  BASEPTR

            IF      @BIGMODEL
BASEPTR     proc    far
            ELSE
BASEPTR     proc    near
            ENDIF

BEGIN:      push    bp      ; save registers (standard entry)
            mov     bp,sp
            pop     ax      ; base pointer to ax as return value
            push    ax
            mov     sp,bp   ; restore registers (standard return)
            pop     bp
            ret             ; return
SCCSID      db      '@(#)baseptr.asm  5.3.0'
BASEPTR     endp
            include epilogue.h          ;C86
            end
```

Listing 4. DEMODBUG.C; Demonstrate debug functions

```
    static char SCCSID[] = "@(#)demodbug.c  5.3.4";
/*------------------------------------------------------*/
/*  SPECIAL DEFINES */
#define     void    int
/*------------------------------------------------------*/
/*  INCLUDE */
#define     EXTERNAL extern
```

Listing continues

```
#include    <stdio.h>
#ifdef      DEBUG_1
#include    <debug.h>
#endif
#undef      EXTERNAL
/*-----------------------------------------------------*/
int SQUARE(P_G)
    int  P_G;                    /* in */

    /* Return square. */
{
    static char FUNC[] = "SQUARE";
    int       RTN;

/* begin */
#ifdef DEBUG_1
    if (DBUGFLG3) {
        DBUGBEGN(FUNC,SCCSID);
        DBUGINT(FUNC,"P_G",P_G);
    }
#endif

    RTN = P_G * P_G;

#ifdef DEBUG_1
    if (DBUGFLG3) {
        DBUGINT(FUNC,"RTN",RTN);
        DBUGEND(FUNC);
    }
#endif

    return (RTN);
} /* SQUARE */
/*-----------------------------------------------------*/
SUMSQ(P_F,P_D,P_E)
    int *P_F;                    /* out */
    int  P_D;                    /* in  */
    int  P_E;                    /* in  */

    /* Compute sum of squares. */
{
    static char FUNC[] = "SUMSQ";
    extern int  SQUARE();

/* begin */
#ifdef DEBUG_1
    if (DBUGFLG2) {
        DBUGBEGN(FUNC,SCCSID);
        DBUGINT(FUNC,"P_D",P_D);
        DBUGINT(FUNC,"P_E",P_E);
        DBUGBIN(FUNC,"P_E",P_E);
    }
#endif

    *P_F = SQUARE(P_D) + SQUARE(P_E);

#ifdef DEBUG_1
    if (DBUGFLG2) {
        DBUGINT(FUNC,"P_F",*P_F);
        DBUGPTR(FUNC,"P_F",P_F);
        DBUGEND(FUNC);
    }
#endif
} /* SUMSQ */
/*-----------------------------------------------------*/
int PASSPARM(P_A,P_B,P_C,P_D,P_E)
    int  P_A;                    /* in */
    int  P_B;                    /* in */
    int  P_C;                    /* in */
    int  P_D;                    /* in */
    int  P_E;                    /* in */

    /* Pass Parameters. */
{
    static char FUNC[] = "PASSPARM";

/* begin */
#ifdef DEBUG_1
    if (DBUGFLG2) {
        DBUGBEGN(FUNC,SCCSID);
        DBUGSTAK(FUNC,7);
    }
#endif

    printf("\n%s: Parameters '%c' '%c' '%c' '%c' '%c'",
        FUNC,P_A,P_B,P_C,P_D,P_E);

#ifdef DEBUG_1
    if (DBUGFLG2) {
        DBUGEND(FUNC);
    }
#endif
    return;
} /* PASSPARM */
/*-----------------------------------------------------*/
main()
{
    static char FUNC[] = "main";
    static int  A = 2;
    static int  B = 3;
    int         C;
    extern int  SUMSQ();

/* begin */
#ifdef DEBUG_1
    DBUGOPEN();
    if (DBUGFLG1) {
        DBUGBEGN(FUNC,SCCSID);
        DBUGSTR(FUNC,"Msg","Compute Sum of Two Squares");
        DBUGINT(FUNC,"A",A);
        DBUGINT(FUNC,"B",B);
        DBUGPTR(FUNC,"C",&C);
    }
#endif

    SUMSQ(&C,A,B);
    printf("\nThe Value of %d**2 + %d**2 Is %d\n",A,B,C);

#ifdef DEBUG_1
    if (DBUGFLG1) {
        DBUGINT(FUNC,"C",C);
        DBUGPTR(FUNC,"C",&C);
    }
#endif

#ifdef DEBUG_1
    if (DBUGFLG1) {
        DBUGSTR(FUNC,"Msg","Pass Parameters for Stack Display");
    }
#endif

    PASSPARM('A','B','C','D','E');

#ifdef DEBUG_1
    if (DBUGFLG1) {
        DBUGEND(FUNC);
    }
#endif

} /* main */
```

Listing 5. DEBUG.LST; Debug output file

```
00:55:42 DBUGOPEN: Debug Started with SCCSID='@(#)debug.c  5.3.8'
00:55:42 DBUGOPEN: DBUGFLG0        =     0
00:55:42 DBUGOPEN: DBUGFLG1        =     1
00:55:42 DBUGOPEN: DBUGFLG2        =     1
00:55:42 DBUGOPEN: DBUGFLG3        =     1
00:55:42 DBUGOPEN: DBUGFLG4        =     1
00:55:42 DBUGOPEN: DBUGFLG5        =     0
00:55:42 DBUGOPEN: DBUGFLG6        =     0
00:55:42 DBUGOPEN: DBUGFLG7        =     0
00:55:42 DBUGOPEN: DBUGFLG8        =     0
00:55:42 main    : Begin with SCCSID='@(#)demodbug.c  5.3.4'
00:55:42 main    : Msg             ='Compute Sum of Two Squares'
00:55:42 main    : A               =     2
00:55:43 main    : B               =     3
00:55:43 main    : C               = Ptr  Segm=6771 Ofst=FF48  Abs=77658
00:55:43 SUMSQ   : Begin with SCCSID='@(#)demodbug.c  5.3.4'
00:55:43 SUMSQ   : P_D             =     2
00:55:43 SUMSQ   : P_E             =     3
00:55:43 SUMSQ   : P_E             =0000000000000011b
00:55:43 SQUARE  : Begin with SCCSID='@(#)demodbug.c  5.3.4'
00:55:43 SQUARE  : P_G             =     2
00:55:43 SQUARE  : RTN             =     4
00:55:43 SQUARE  : End
00:55:43 SQUARE  : Begin with SCCSID='@(#)demodbug.c  5.3.4'
00:55:43 SQUARE  : P_G             =     3
00:55:43 SQUARE  : RTN             =     9
00:55:43 SQUARE  : End
00:55:43 SUMSQ   : P_F             =    13
00:55:43 SUMSQ   : P_F             = Ptr  Segm=6771 Ofst=FF48  Abs=77658
00:55:43 SUMSQ   : End
00:55:44 main    : C               =    13
00:55:44 main    : C               = Ptr  Segm=6771 Ofst=FF48  Abs=77658
00:55:44 main    : Msg             ='Pass Parameters for Stack Display'
00:55:44 PASSPARM: Begin with SCCSID='@(#)demodbug.c  5.3.4'
00:55:44 PASSPARM: Stack            Address    Hex     Int  Char

                                    6771:FF38  FF4A   -182  ^!
                                    6771:FF3A  02B0    688  ^*
                                    6771:FF3C  5AB0  23216  ^*
                                    6771:FF3E  0041     65  A
                                    6771:FF40  0042     66  B
                                    6771:FF42  0043     67  C
                                    6771:FF44  0044     68  D
                                    6771:FF46  0045     69  E
                                    6771:FF48  000D     13  ^M
                                    6771:FF4A  FFF2    -14  ^!
00:55:44 PASSPARM: End
00:55:44 main    : End
```

End Listing 5

VM/386

FOR ORDERING AND INFORMATION CARD
SEE INSERT CARD.

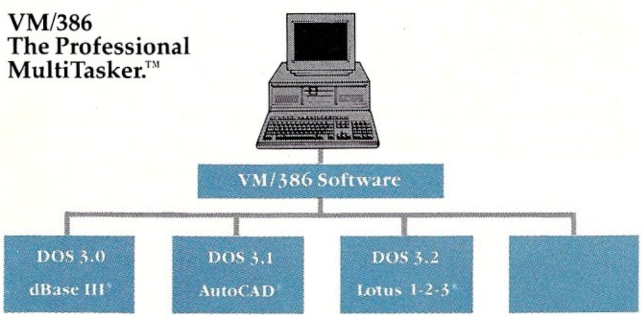
IT'S LIKE HAVING A DESKTOP FULL OF PCs

We know why you have it. Power. And now you can multiply that 386 power. Through true multitasking. With VM/386.™

VM/386 is the 80386 control program that brings you true DOS multitasking. VM/386 uses the virtual 8086 mode, built into the 80386 processor, to create individual virtual machines. You can load a different application in each virtual machine. It's like having a desktop full of PCs.

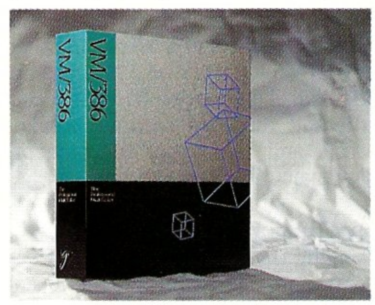
You have complete control over the virtual machines. You can tailor each virtual machine to fit your needs—and priorities. Each virtual machine has its own DOS, CONFIG.SYS, AUTOEXEC.BAT, and memory-resident programs along with its application. And each virtual machine is isolated from the others. A malfunction in one program doesn't destroy the others.

**VM/386
The Professional MultiTasker.™**



There is virtually no limit to the number, or kind, of applications that can run simultaneously. For example, you can recalculate a 1-2-3® spreadsheet, sort a dBASE III® file, and receive your E-mail—all at the same time. You can even run more than one AutoCAD™ program simultaneously. And EGA applications run in the background as well as in the foreground—perfectly.

VM/386 protects your investment in software—and hardware. VM/386 will run all DOS software. Existing and future. Without modification. VM/386 is compatible with 80386-based PCs—either native or Intel® Inboard 386.™ VM/386 will support the monitors, hard disks, printers, mice, modems—even the network—you're currently using—or plan to use.



With VM/386 you use familiar commands. You don't have to buy new software or upgrades to get true multitasking. No PIF files. No special loaders. VM/386 works the way you do. Only lots faster.

VM/386 is easy to install. Easy to learn. And easy to use.

Dramatically increase your productivity. Start using all the power built into your 80386. Phone or write today.

IGC
4800 Great America Parkway
Santa Clara, California 95054
(408) 986-8373

System Requirements

80386-based computer such as COMPAQ® DESKPRO® 386® or 80286 computer with Intel® Inboard™ 386

One 1.2 Mb (5¼") disk drive or, one 3½" microfloppy

One hard disk drive

DOS 3.0 or later

2 Mb memory recommended

Not copy protected

Package includes both 5¼" and 3½" media.

VM/386 is a trademark of IGC.
AutoCAD is a trademark of Autodesk Inc.
COMPAQ and COMPAQ DESKPRO 386 are registered trademarks of Compaq Computer Corporation.
dBASE III is a registered trademark of Ashton-Tate Corporation.
Intel is a registered trademark of Intel Corporation.
Inboard is a trademark of Intel Corporation.
1-2-3 is a registered trademark of Lotus Development Corporation.

IGC

The Periscope Debuggers

A hardware/software debugging family that makes life easier for C programmers.

by Joseph A. Sabin, Jr.

The Periscope line of debuggers includes the Models I, II, II-X, and III. The Model I consists of a plug-in card with 56K of protected memory, a breakout switch and software. The Model II consists of a breakout switch and software. The Model II-X has only software. The Model III has a 64K-protected memory card, a breakout switch, and the ability to trace program execution in real time.

Installation

Installing the Model I is relatively easy and requires inserting a plug-in card and attaching the breakout switch to the card.

The Periscope Model II-X requires only the installation of the software. The Model II adds the installation of the breakout switch and requires you to insert a pin into a bus slot along with an existing plug-in card. This is not difficult to do on a standard PC/AT or PC/XT system or compatible; however, to accomplish this feat on a Compaq portable, you need to be a contortionist.

In addition to the installation of the plug-in card and breakout switch, the Model III requires that you remove the 8087 or 80287 coprocessor, if one is installed, and fit a socket with a ribbon cable into the now empty socket and place the coprocessor into the taller socket. Then the ribbon cable is connected to the plug-in card.

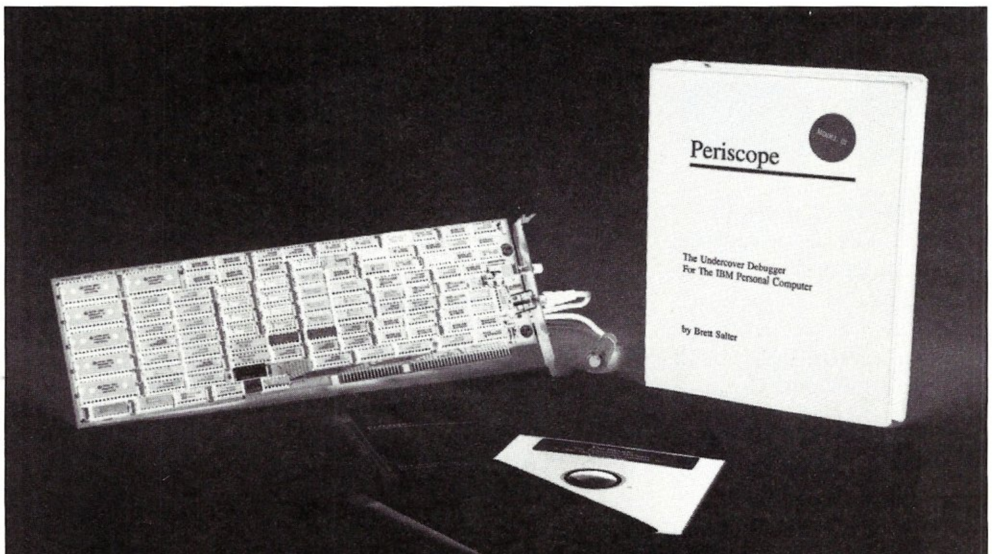
Once the Periscope debugger software is installed and your AUTOEXEC.BAT file is modified (advisable but not required), you must sit down and read the rather complex manual.

Using The Debugger

If you are writing an operating system add-in, a pop-up, or a memory-resident program, the Model I or III with protected memory is the preferred tool. Their protected memory is needed because, if you are not careful, these types of programs can crash into disk reads or writes, memory allocation calls, or other operations. The ability to call the debugger when the memory in your system has been totally scrambled is, without question, a very valuable debugging feature.

Under ordinary conditions, straightforward programming (even a C program with heavy use of pointers) will not crash the memory area holding the resident Periscope debugger software, which allows the programmer to use a software-only debugger like the Model II-X. It gives the programmer debugging capability similar to other full-featured symbolic debuggers, such as Microsoft's Codeview. However, if the software you are developing crashes the system and Ctrl-Alt-Del will not work, you need a hardware breakout switch. If you fit this class of programmer, then I recommend you avoid the II-X software-only model. Most programmers will find the Model II almost as good as the protected memory models, and more cost effective.

The Models II and III memory boards provide a safer place to stick debugger software and allow you to have 56K–64K more memory available for your own code. In my system, this is important because otherwise I can't have all my resident software in memory at one time. Without the pro-



tected memory, the Periscope software competes for the same memory and it does not all fit. Periscope is not something you load as you need it—if it's not there initially it's useless.

The Periscope debugger can show source code as you single-step through execution, demonstrate selected variables, and show you what is in the memory locations that they occupy. If they are pointers, you can see the value at the location to which they point. For example, if your pointer is pointing at one of DOS's low-memory locations to show you what's there, it should (generally speaking) not be used for storage since it will probably trash DOS. I was amazed that a pointer in one of my programs pointed to the wrong place and still worked—it was a bomb waiting to happen, I found it with Periscope!

Periscope's debugging software is clear, concise, and seemingly bomb proof. In testing for this review, I formatted disks, used WordStar, compiled a Turbo Pascal program to disk, loaded programs, saved programs, and made life as difficult as possible for the Periscope debugging software. Pressing the Periscope breakout button never crashed the system and I was always able to go back to what I was doing as if nothing happened. Now that is impressive!

Having the breakout switch is worth the price, if only to prevent having to turn the computer off and on when a runaway pointer trashes everything in memory, leaving you with only the ROM BIOS and a blinking cursor. Once my software did cause a crash from which the Periscope Model II could not reboot the computer forced me to turn the computer off and on to regain control. The Models I and III, however, never allowed the system to go into hyperspace. Their protected memories always allow the debugger to retain its composure and help me whenever I ask for it. A good friend indeed.

As with all software, I only use some of its power, but I can say with confidence that it works for almost any debugging task, if you choose the correct version for your work.

Usually, I comment on how helpful the technical support for the product is, but this time I did not find it necessary to call for tech support. Okay, so I feel guilty for not calling just to find out, but I don't think most programmers will need to call either.

The manual covers everything you need to know (not always clearly, but it's there). It consists of 11 chapters, an appendix, and an index. The index is clear and covers everything that I could think to look for. However, additional examples would help in teaching uses for this sophisticated programming tool. If you program in Assembler or C, there is sufficient information to get you started and you will learn as you go, using the manual mostly as a reference. The disk contains the source code for the programming samples in the manual (yes, they are the same) and this provides powerful tutorial for the experienced programmer.

Summing Up

If you program in C or assembler under MS-DOS, then the Periscope family of debuggers will make your life easier. Compared to other debuggers, they are more useful in discovering real-

> *I can say with confidence that it works for almost any debugging task, if you choose the correct version.*

time errors. I have not used other hardware-based debuggers, but compared to what I am used to doing to find a bug, this is worlds apart. Based on cost alone, the Periscope line has a very good price/performance ratio. I don't imagine that another debugger approach could offer much more. □

Joseph A. Sabin, Jr., is Director of Systems Development for New Hope Communications, Inc. a national magazine publisher. He works primarily with accounting and fulfillment software programmed in C under PC-MOS and dBASE III+ under MS-DOS.

Product Information

Periscope I	$345.00
Periscope II	$175.00
Periscope II-X	$145.00
Periscope III	$995.00 (8 MHz)
	$1,095.00 (10 MHz)

The Periscope Company, Inc.
1197 Peachtree St., Plaza Level
Atlanta, GA 30361
(404) 875-8080

by P. L. Olympia, Ph.D.

dBASE Index Files

Anyone who has ever used dBASE knows that the three primary commands used to locate database records are FIND, SEEK, and LOCATE. The first two find records in just a few seconds, regardless of database size, while the third takes almost forever if the file is large enough. The difference in speed is due to record indexing. LOCATE reads each database file record sequentially until a match is found, while SEEK and FIND use a previously created index file to point to the desired record. Clearly, you should use index files whenever possible, but it is easy to get carried away and forget that they can be a hindrance in certain situations. For example, if you have to use the LOCATE command (there are occasions when you don't have a choice) you are better off without any active index files.

Better Use of Index Files

A common mistake that beginners make is to have too many open index files, especially during repetitive append and edit operations, because the application tends to get bogged down as each open index file is updated. Although dBASE allows up to seven index files to be open per work area, it is difficult to imagine a situation where that many index files are needed for one database, or where that many index files must be active at one time. Unless you run on a network, you might consider the advantage in speed gained by batch appending/editing without an active index file, then indexing when you are done. The method you should use depends on how many records are being changed at one time, and how many index files need to be updated. When running on a network, indexing requires either a file lock or exclusive use of a file and should be avoided whenever possible.

Another common mistake made by dBASE users is to devise index expressions that are unnecessarily lengthy or complicated. I have seen many name/address dBASE applications index a database using the key expression LASTNAME+FIRSTNAME. That not only wastes a lot of disk space, but does not optimize record searches. Since it is difficult to get a consensus on how long a name field should be, many programmers play it safe by assigning a field width of 20–25 characters to LASTNAME and FIRSTNAME. In practice, very few names require so many reserved characters, so you end up with wasted space and an index file that is less than optimal.

A better expression would be something like TRIM(LASTNAME) + TRIM(FIRSTNAME). Still, do you really need the whole name? I prefer to use a different expression such as SUBS(LASTNAME,1,8) + SUBS(FIRSTNAME,1,4), since most last names should differ by the time you get to the ninth character. If two names are still the same at that point, I don't need more than four characters of FIRSTNAME. You may disagree with the length of the substrings I have chosen, but you really don't need the whole name.

Creative Index Expressions

As these examples show, an index expression need not be limited to the fields in your database. You may use dBASE functions, constants, and even memory variables. However, using a memory variable in an index expression is risky business because you may not be able to retrieve desired records if the memory variable changes value or becomes undefined from one session to the next.

One of the most versatile functions in dBASE is the IIF (Immediate IF) function. It is often used in report writing, but is just as useful as part of an index expression. For a quick introduction to IIF, consider the following code fragment:

```
IF mvar = "PRN"
   mprint = .T.
ELSE
   mprint = .F.
ENDIF
```

Those five lines of code can be reduced to one, courtesy of the IIF function:

```
mprint = IIF(mvar = "PRN",
   .T., .F.)
```

(Actually, due to the special nature of the code fragment shown above, you could also reduce it to one line even in dBASE II using STORE mvar = "PRN" to mprint.)

Suppose your database of addressees contains two zip codes, one for a business address and one for a home address. You prefer to index on BUS_ZIP, but if it is missing, you'd settle for HOME_ZIP. Your index expression simply becomes:

```
IIF(bus_zip > [], bus_zip,
   home_zip)
```

There is one thing to keep in mind when you use functions like IIF in an index expression; the function must always return a value with the same length. For that reason, we may not use, say, CITYNAME as a substitute for HOME_ZIP in our zip code example.

Index In Descending Order

You may have been surprised to discover that dBASE III Plus does not have a built-in facility to index in descending order of key value. This oversight has caused a lot of confusion, particularly when even "gurus" sometimes recommend workaround expressions that are unnecessarily cumbersome.

First, consider the case of numeric fields. If we want to index in descending order of a numeric field, all we need is an expression that involves subtracting the field from a large numeric constant. The result is a number that gets smaller as the value of the field gets larger. If the numeric field is *nf*, the index statement would be something like:

```
INDEX ON 999999 - nf TO
   <ndxfile>
```

or better yet

```
INDEX ON -nf TO <ndxfile>
```

Now, consider the case of date fields. Most people will tell you that to index in descending order of dates, you must convert the date value in the form YYYYMMDD and subtract that from a large number. If the date field is *df*, the index statement they recommend reads like it came from an algebra textbook:

```
INDEX ON STR(9999999 -
      YEAR(df)*10000 +
      MONTH(df)*100
      + DAY(df)),8)
      TO <ndxfile>
```

There is a much easier way. Note that, like most software, dBASE treats a date variable as a number. We know

Figure 1. A Sample Output from DBNDXPO

```
C>dbndxpo d:\darwin\*.??X

Filename      Type  Index Expression
-----------   ----  ------------------------------------
ADB2.NDX      dB2   !($(NAME,1,5)+STATE
FXXFINR.NDX   dB3   projnum+fy
FOXY.IDX      FOX   SUBS(lastname,1,8)+SUBS(firstname,1,4)
CLIPME.NTX    CLIP  d_code+d_name
MIPS.FOX      Not an index file
```

that from the way dBASE stores dates in its files. We also know that from the fact that you can do arithmetic with dates. For example,

```
date1 + number1 = date2
date1 - number1 = date3
date1 - date2   = number1
```

Theoretically, you could add two dates but the result would be nonsensical, so dBASE gives you a "Type mismatch" error.

Since we can, with few exceptions, treat dates as numbers, indexing in descending order of dates is really quite simple. We can't have an expression

```
INDEX ON -df TO <ndxfile>
```

but if we subtract *df* from a small enough "number," we have what we want and we don't have to be algebra nuts to understand it:

```
INDEX ON CTOD("01/01/00") -
   df TO <ndxfile>
```

OOPS, I Can't Recall the Key Expression

There you are, building your dBASE application system masterpiece, creating index files left and right. Two weeks later, you are staring at a directory full of NDX files, and you can't remember the key expression used to build them. Of course, if you remember which NDX files go with which DBF files, you can go into dBASE, use the files one at a time, and display the key expression you've forgotten with the help of dBASE's DISPLAY STATUS command, but that's a lot of work.

What you need (other than a good memory) is a program that can be run from DOS that will read every NDX file you have on disk and report the key expressions. Enter DBNDXPO, a program I wrote some time ago. This small program displays the key expression of index files created by dBASE III (and Quicksilver), dBASE II, Clipper and FoxBASE+. Figure 1 shows a sample invocation and output.

The program is primitive and is based on empirical data obtained by creating a few index files and examining their structure with DEBUG. I knew that each index file has a header record, and that the key expression is stored somewhere in that record. By experimentation, I found that the expression began at the following locations in the file:

Software	Byte Offset
dBASE II	10
FoxBASE+	16
Clipper	22
dBASE III	24

Naturally, the program will produce erroneous results if any of the file structure changes. However, the program works as expected, as long as the key expression is longer than one character. In fact, it is very handy for getting the key expressions of index files into the system documentation, and in making sense of all those index files you have on disk.

Which NDX File Belongs to Which DBF File?

dBASE application systems can use a hundred database files and several hundred index files. When looking at a directory full of database and index files, how do you tell which NDX files are associated with which DBF files? Normally, you can't because of a deficiency in dBASE; there is nothing in an NDX file that identifies the DBF file with which it is associated. dBASE should have placed a tag on an index file as soon as it is created that would carry the name of the associated DBF file. An internal function, say WHICHDBF( ), would then return the tag stored in an NDX file, telling you the name of the DBF file to which it belongs. Unfortunately, there is no such tag.

> *DBXDXPO is very handy for getting the key expressions of index files into the system documentation and in making sense of all those files.*

As it turns out, Curtis Hoffman has a system that essentially does the same thing. Curtis has a shareware file called DB_DBUG2.ARC available from many bulletin board systems. DB_DBUG2 contains eight dBASE utility programs, along with their Turbo Pascal source code. Two of those programs, INSERT and PICKIT, depend on the fact that the header (anchor node) record of a dBASE NDX file contains a lot of unused space, specifically byte offset 124–511. INSERT writes the name of a DBF file beginning at offset 496 of the NDX file header record. Later, if you want to know the name of a DBF file associated with an index file, you run PICKIT, which simply reads the tag deposited by INSERT and displays it to you.

This approach can use improvement in two areas. First, INSERT and PICKIT do not accept wildcards. If you have to tag five index files with the same DBF name, you must run INSERT five times. Second, the two files could be combined into one, thereby saving disk space and leaving one less file to deal with. I wrote WDBF (short for WHICHDBF) to correct these two deficiencies in Curtis' program. WDBF is a C program that is faster and much smaller than either INSERT or PICKIT. It is a free program that also is available from most BBSs. □

P. L. Olympia, Phd, better known as "Dr. dBASE," is a scientist with a doctorate in Chemical Physics. He is coauthor of the book, dBASE Power: Building and Using Programming Tools, *recently published by Ashton-Tate.*

by Michael Cherry

New LAN Products To Support PS/2 and Twisted-Pair Media

This month we will look at new product developments in two areas: PS/2 Micro Channel network interface cards and twisted-pair media support.

IBM's Token-Ring Adapter A Network Interface Cards

IBM has introduced the Token-Ring Adapter A network interface cards for use with its new PS/2 Micro Channel-based systems. Many users first tried to use these cards with Novell NetWare in the same manner that they had used the IBM Token-Ring Adapter I and II cards, i.e., they first loaded TOKREUI.COM and then tried to load the NetWare shell ANET3.COM. This generally caused the PS/2 workstation to hang.

The Token-Ring Adapter A cards will not work with TOKREUI. Instead, they require the IBM PC LAN Support Program. This program must be loaded first, then load the NetWare shell to use the PS/2s as workstations on a Novell network.

Since IBM introduced the PS/2 Micro Channel systems, LAN users have been searching for Micro Channel network interface cards other than IBM's PS/2 Token-Ring Adapter A cards. PS/2 Micro Channel network interface cards for other LAN topologies also have been introduced. Currently, there are cards for ARCnet and Ethernet (*à la* 3Com), and more are on the way. However, these cards must also be installed carefully.

Pure Data Network Interface Cards

The first Micro Channel ARCnet network interface card, the PDIuC508 card, was introduced by Pure Data. Pure Data has always had an excellent reputation in the ARCnet arena, and they deserve a lot of praise for getting this excellent Micro Channel card to market so early.

Micro Channel cards work quite differently from their XT and AT counterparts. Most notably, they lack jumpers or switches on the cards, so all interrupts settings and other addressing must be set via software.

The first step in working with Micro Channel cards is to copy the drivers from the interface card diskette to the backup of the PS/2 Reference Disk. This will allow the PS/2 to recognize the card and accept the setup. During setup, you will set an interrupt level, base memory address, and base I/O address. In addition, it is necessary to set a node address. When you first bring up the setup program, the node address will be at 128, and you can use the appropriate keys to increase or decrease the desired node address.

The current Novell shells that are used with Pure Data or Standard Microsystems ARCnet network interface cards will work in PS/2 Models 50 and 60. They will, however, not work in a Model 80, but Pure Data can supply an ARCnet shell that will work with the Model 80. Novell also has shells that can be used with DOS 3.3 when run on PS/2 Micro Channel workstations.

No doubt there will be other Micro Channel ARCnet cards available in the near future, but Pure Data has already proven that this card is reliable and workable in the Micro Channel environment.

In addition to having the first Micro Channel ARCnet network interface card, Pure Data also has come up with the first ARCnet network interface card, the PDTC508, that fits into Toshiba T1100, T1200, and T3100 portable microcomputers. This means that you can connect a Toshiba portable as a node on an ARCnet network. Now, Toshiba portable users have an alternative to floppy disks or a serial communications link to transfer data between the portable and the LAN.

3Com Etherlink MC

3Com has developed a PS/2 Micro Channel interface card for use on 3Com Ethernet local area networks. Again, there is an Adapter Description File that works with IBM's Programmable Option Select utility to ease installation and configuration. There are no hardware jumpers to set; interrupt level, I/O and memory addresses, and external or internal (DIX/BNC) transceiver operation are software-selectable. These cards work with the 3+ network operating system or other NetBIOS-compatible operating system.

As of this writing, 3Com's new Etherlink interface cards will not work with Novell NetWare shells. Novell has indicated that they are working on new NetWare shells for use with Etherlink Micro Channel network interface cards.

Standard Microsystems ARCnet on Twisted-Pair

Regular readers of this column know that we at HallComm NetWork Services are not fans of twisted-pair cable for LANs. However, we feel that new products are making this type of cabling more viable.

One of these products is the new twisted-pair ARCnet network interface card from Standard Microsystems. This card fits in an expansion slot in any IBM PC, XT, AT, or compatible. Workstations with these cards can be connected together on a single twisted-pair segment in either a daisy-chain or multidrop configuration. The daisy-chain or multidrop segment can extend up to 400 feet.

Expansion is permitted by connecting twisted-pair segments with a two-port twisted-pair repeater, and a twisted-pair network can be bridged to a coax network with a twisted-pair link. A typical system configuration is shown in Figure 1.

3Com Ethernet on Twisted-Pair

3Com has three products that provide 10 mbps Ethernet operation on unshielded twisted-pair cable. The three products are:

- PairTamer
- Multiconnect
- LanScanner.

PairTamers are adapters that replace the existing modular phone jack. Both the phone and the workstation are plugged into the PairTamer, allowing the voice and data signals to coexist on the cable system without interference. Don't confuse the PairTamer with a simple balun (impedance match-

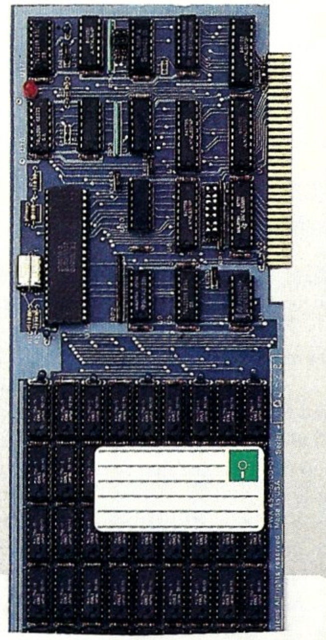
. . . *continued on page 67*

Think fast! Pick the better fit...

Our 5th Year Bonus!
Mention this ad when ordering and get your choice of a V 20-8 (replaces 8088) or $20 off on your Battery Backup purchase!

FLOPPY DISK.

- Fills time between coffee breaks
- Makes a hard disk seem *fast*
- Your computer appears busy (even if you aren't!)
- Wears out moving parts



SEMIDISK Disk Emulator.

- Gets that job done NOW
- Makes a hard disk seem *slow*
- Maximizes your productivity with anything from databases to compilers
- Totally silent operation

...for YOUR demanding tasks.

SURPRISE! *Neither* is memory mapped, so they don't affect your precious Main Memory. *Both* retain data indefinitely - even with the computer turned off.

THE SEMIDISK SOLUTION. You could invest in a series of "upgrades" that turn out to be expensive band-aids without solving your real problem. Even those "Accelerator" and "Turbo" boards do little to speed up disk-bound computers. If your applications spend too much time reading and writing to disk (and whose don't?), you won't want to settle for anything less than a SemiDisk disk emulator. The SemiDisk comes in 512K and 2Mb capacity. More boards may be added to make up to an 8 Megabyte SemiDrive!

SPEED THAT'S COMPATIBLE. PC, XT or AT, if you need speed, the SemiDisk has it. How fast? Recent benchmarks show the SemiDisk is from 2 to 5 times faster than hard disks, and from 25% faster (writing) to several times faster (random reads) than VDISK and other RAMdisk software that gobble up your main memory.

MEMORY THAT'S STORAGE. Using our small external power supply, with battery backup, your data remains intact through your longest vacation or even a seven-hour power failure!

CELEBRATE WITH US! Now, SemiDisk celebrates its fifth birthday with a special offer for IBM-PC owners. Buy a SemiDisk now and we'll include an 8 MHz V-20 microprocessor (replaces the 8088) to make your new SemiDisk run even faster. Don't need the V-20? We'll take $20 off the price of your Battery Backup Unit!

	512K	2Mbyte
IBM, PC, XT, AT	**$495**	**$ 795**
Epson QX-10	**$495**	**$ 995**
S-100 SemiDisk II	**$795**	**$1295**
S-100 SemiDisk I	**$299**	-------------
TRS-80 II, 12, 16	**$495**	**$ 995**
Battery Backup	**$130**	**$ 130**

Someday you'll get a SemiDisk.
Until then, you'll just have to...wait.

SemiDisk

MasterCard VISA

SemiDisk Systems, Inc., 11080 S.W. Allen Blvd., Beaverton, Oregon 97005 (503) 626-3104

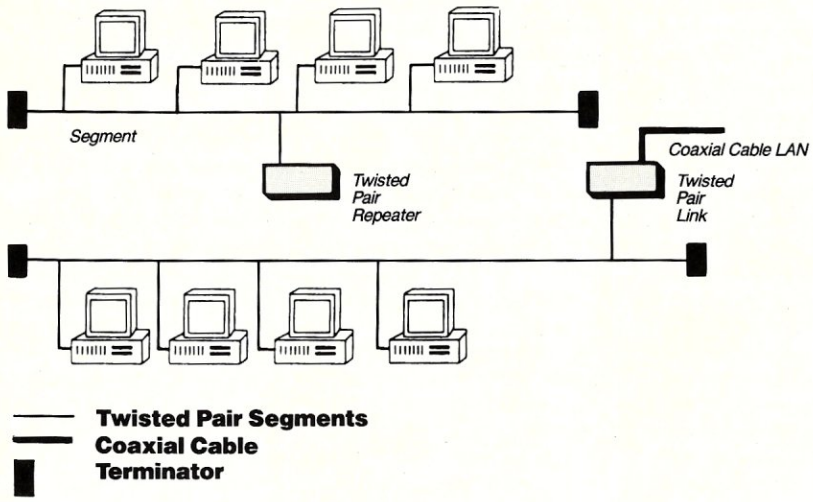


Figure 1. A typical ARCnet twisted-pair network

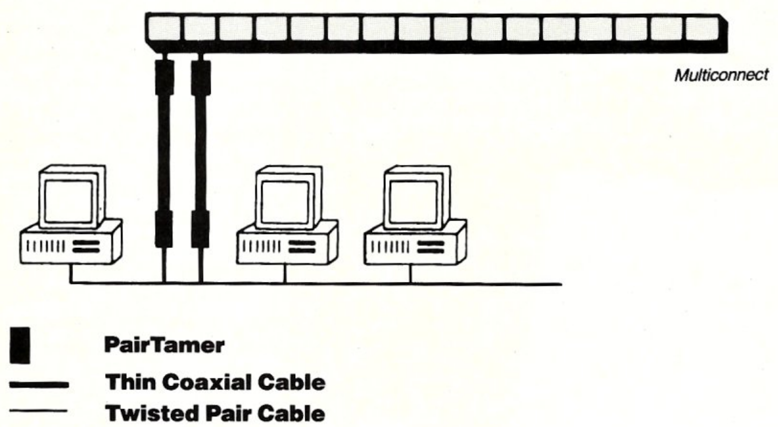


Figure 2. A typical 3Com twisted-pair network

Continued from page 64
ing transformer). We have examined the electronics of the PairTamer and it is more then a balun, as is reflected in the cost.

The Multiconnect is a repeater with slots for 15 modules. Any combination of coaxial, twisted-pair, or fiber-optic wiring can be used by selecting the appropriate module.

The LanScanner is a hand-held, battery-powered test instrument that enables cable technicians and network administrators to measure twisted-wire cable characteristics and determine whether or not a cable system can support data transmissions. A typical 3Com twisted-pair network is shown in Figure 2. Note that 3Com allows you to go from the twisted-pair, via the PairTamer to a segment of coaxial cable.

Anything Goes?

Does this mean that anything goes—can any twisted-pair cable be used? The answer is no! Both Standard Microsystems and 3Com have published the technical specifications for the twisted-pair cable system, and these must be met. The specifications are available from the manufacturer and should be checked prior to deciding to use twisted-pair wiring. In the case of 3Com, you will want to use the LanScanner to test the existing twisted-pair wiring to determine its suitability for use with 3Com equipment. □

Michael Cherry is Vice President of Technical Support for HallComm Network Services (HNS), a company devoted exclusively to designing and implementing LAN systems.

Product Information

HallComm NetWork Services
8101 E. Prentice Ave., Ste. 304
Englewood, CO 80111
(303) 770-6387

Pure Data
1740 S. I-35, Ste. 140
Carrollton, TX 75006
(214) 242-2040

Standard Microsystems Corporation
35 Marcus Blvd.
Hauppauge, NY 11788
(516) 273-3100

3Com
3165 Kifer Rd.
Santa Clara, CA 95052-8145
(408) 562-6400

New Products

When contacting vendors, please mention that you read about their products in Micro/Systems.

Manufacturers who would like to have their products listed here should send their new product releases to the Managing Editor, Micro/Systems, *501 Galveston Drive, Redwood City, CA 94063.*

PC-Compatible Products

Card Offers Bus Analysis

Applied Physics, Inc., has released the BusTrak Microsystem Bus Analyzer for microcomputer diagnostics and hardware and software development. Unlike many other test systems, A single XT/AT BusTrak card combines many of the features of logic analyzers, code debuggers, and PC testers. The card can be inserted into the development system, or it can be operated from a second system to perform diagnostics or advanced post processing. BusTrak has a window-based menu for easy operation and will support a mouse. The BusTrak also can capture from 8,000 to 32,000 bus states in real time, depending on the computer model. Unlike software debuggers, BusTrak captures all bus signals. Flexible triggering allows the user to set any breakpoint condition. Options include address or address range, data bus, I/O or memory access, reads or writes, and combinations. Post processing of captured data offers an alternative to scrolling through screens or bus states. The user shell can set the captured data for you and find specific events.

The XT/AT BusTrak is scheduled for release in the first quarter of 1988. Additional versions for IBM's PS/2 and the Macintosh II are scheduled for release in the second quarter. Prices range from $1,500 to $2,500, depending on the model. For more information, contact **Applied Physics, Inc.**, Purdue Research Park, Lafayette, IN 47907; (317) 497-1718.

Tape Backup System Taps PS/2 Multitasking

Maynard Electronics, Inc., has begun shipping the MaynStream 60, a high-performance, 60-megabyte tape backup system for the IBM PS/2 models 60 and 80. The system includes the drive, adapter card, cables, data cassette, and control software. The MaynStream 60 fits into the 5¼-inch opening of the Model 60 or 80, leaving room for a 5¼-inch hard disk. Both the adapter and the software take advantage of the Micro Channel bus architecture, including multitasking to allow simultaneous reading from the hard disk while writing to tape, with automatic reading after writing to assure the accuracy of each backup. MaynStream's software supports Novell, 3Com, IBM PC Network, and Token-ring LANs with remote backup/restore of a file server from any workstation, including file-by-file backups without shutting down the network.

Maynard offers a variety of PS/2 backup configurations, including 20 MB ($1,395) or 60 MB ($1,695) portable cassette systems, and 60 MB ($2,095) or 150 MB ($2,295) portable ¼-inch cartridge tape backup systems. The internal 20 MB ($995) cassette system and 150 MB ($1,995) cartridge system can mount inside a Model 60 or Model 80.

For more information, contact **Maynard Electronics, Inc.**, 460 E. Semoran Blvd., Casselbury, FL 32707; (305) 331-6402.

Other Hardware Products

M-Test Offers Low-priced Serial Breakout Box

M-Test Equipment has released its Model 225, a high-quality RS-232C serial breakout box that includes parallel interface testing capabilities. The Model 225 has 52 LEDs that give 4 state signal indications. Twenty-six in-line switches and 52 sockets allow breaking and redirection of 25 lines plus one unassigned line. A battery simulates high or low signals.

The Model 225 from M-Test

The Model 225 sells for $229. For more information, contact **M-Test Equipment**, P.O. Box 146008, San Francisco, CA 94114; (415) 864-1076.

New Software Products

New Optimizing Compiler Produces Tight Code, Fast

Watcom Group, Inc., has just released C6.0, an optimizing C compiler and development system that is ideally suited for large memory models and floating-point computation. The compiler and development systems also permits extensive fine-tuning with a variety of user options , such as in-line substitution of machine code for performance-critical areas. Watcom C6.0 supports small, medium, compact, large, and huge memory models, and NEAR, FAR, and HUGE keywords. Dhrystone tests on a PC reveal that C6.0 will generate 992 bytes in 89 seconds on a small memory and 1001 bytes in 18 seconds. Included in the C6.0 package is a full C optimizing compiler, a source editor, full-screen source-level debugging, a full ANSI runtime library, an overlay lnker, an object librarian, MAKE, and a disassembler. The package includes Express C, also available separately, which provides a seamless development environment with diagnostics. Express C has error checking that uncovers bugs in the "correct" code, and includes MAKE, a disassembler, an object librarian, and an overlay linker.

Watcom's C6.0 has a list price of $495 and Express C retails for $125. Introductory discounts are available for a limited period. For more information, contact **Watcom**, 415 Philip St., Waterloo, Ontario, N2L 3X2, Canada; (519) 886-3700. □

Classifieds

Micro/Systems Journal *accepts Classified Ads. The charge is $6/line (3 lines minimum, 7 lines maximum); 40 characters max./line. Three times frequency $15/line; six times $25/line; non-profit clubs $2/line. Logos, special type, etc. are extra charge. Check must accompany ad copy. Send to M & T Publishing Inc., 501 Galveston Dr., Redwood City, CA 94063.*

WE WELCOME YOUR ARTICLES

We are always glad to hear from potential authors who have an interesting tale to tell. If you are interested in contributing an article that relates to local area networks, multiuser systems, or computer programming, please contact us. For example, in upcoming issues we plan to discuss:

- PC Multiuser Operating Systems
- UNIX on the PC
- Graphics on the PC
- 386 Software Development Tools
- Database Options
- Troubleshooting Local Area Networks
- Modem Standards and Compatibility
- High-Capacity Information Storage

We would welcome your contributions on these and related topics. Please contact:

Tom Woolf
Managing Editor
Micro/Systems
501 Galveston Drive
Redwood City, CA 94063
(415) 366-3600

Advertiser Index

C CODE FOR THE PC

source code, of course

C Source Code

Bluestreak Plus Communications (two ports, programmer's interface, terminal emulation)	$400
CQL Query System (SQL retrievals plus windows)	$325
GraphiC 4.1 (high-resolution, DISSPLA-style scientific plots in color & hardcopy)	$325
Barcode Generator (specify Code 39 (alphanumeric), Interleaved 2 of 5 (numeric), or UPC)	$300
Greenleaf Data Windows (windows, menus, data entry, interactive form design)	$295
Aspen Software PC Curses (System V compatible, extensive documentation)	$250
Vitamin C (MacWindows)	$200
Essential resident C (TSRify C programs, DOS shared libraries)	$165
Essential C Utility Library (400 useful C functions)	$160
Essential Communications Library (C functions for RS-232-based communication systems)	$160
Greenleaf Communications Library (interrupt mode, modem control, XON-XOFF)	$150
Greenleaf Functions (296 useful C functions, all DOS services)	$150
OS/88 (U**x-like O/S, many tools, cross-development from MS-DOS)	$150
Turbo G Graphics Library (all popular adapters, hidden line removal)	$135
American Software Resident-C (TSRify C programs)	$130
CBTree (B+tree ISAM driver, multiple variable-length keys)	$115
PC/IP (CMU/MIT TCP/IP implementation for PCs)	$100
B-Tree Library & ISAM Driver (file system utilities by Softfocus)	$100
The Profiler (program execution profile tool)	$100
Entelekon C Function Library (screen, graphics, keyboard, string, printer, etc.)	$100
Entelekon Power Windows (menus, overlays, messages, alarms, file handling, etc.)	$100
Wendin O/S Construction Kit or PCNX, PCVMS O/S Shells	$95
QC88 C Compiler (ASM output, small model, no longs, floats or bit fields, 80+ function library)	$90
JATE Async Terminal Emulator (includes file transfer and menu subsystem)	$80
MultiDOS Plus (DOS-based multitasking, intertask messaging, semaphores)	$80
ME (programmer's editor with C-like macro language by Magma Software)	$75
WKS Library (C program interface to Lotus 1-2-3 program & files)	$65
Quincy (interactive C interpreter)	$60
EZ_ASM (assembly language macros bridging C and MASM)	$60
PTree (parse tree management)	$60
HELP (pop-up help system builder)	$50
Multi-User BBS (chat, mail, menus, sysop displays; uses Galacticomm modem card)	$50
Heap Expander (dynamic memory manager for expanded memory)	$50
Make (macros, all languages, built-in rules)	$50
Vector-to-Raster Conversion (stroke letters & Tektronix 4010 codes to bitmaps)	$50
Coder's Prolog (inference engine for use with C programs)	$45
C-Help (pop-up help for C programmers ... add your own notes)	$40
Biggerstaff's System Tools (multi-tasking window manager kit)	$40
CLIPS (rule-based expert system generator, Version 4.0)	$35
TELE Kernel (Ken Berry's multi-tasking kernel)	$30
TELE Windows (Ken Berry's window package)	$30
Clisp (Lisp interpreter with extensive internals documentation)	$30
Translate Rules to C (YACC-like function generator for rule-based systems)	$30
6-Pack of Editors (six public domain editors for use, study & hacking)	$30
ICON (string and list processing language, Version 6 and update)	$25
LEX (lexical analyzer generator)	$25
Bison & PREP (YACC workalike parser generator & attribute grammar preprocessor)	$25
AutoTrace (program tracer and memory trasher catcher)	$25
C Compiler Torture Test (checks a C compiler against K & R)	$20
Benchmark Package (C compiler, PC hardware, and Unix system)	$20
TN3270 (remote login to IBM VM/CMS as a 3270 terminal on a 3274 controller)	$20
A68 (68000 cross-assembler)	$20
List-Pac (C functions for lists, stacks, and queues)	$20
XLT Macro Processor (general purpose text translator)	$20

Data

WordCruncher (text retrieval & document analysis program)	$275
DNA Sequences (GenBank 52.0 including fast similarity search program)	$150
Protein Sequences (5,415 sequences, 1,302,966 residuals, with similarity search program)	$60
Webster's Second Dictionary (234,932 words)	$60
U. S. Cities (names & longitude/latitude of 32,000 U.S. cities and 6,000 state boundary points)	$35
The World Digitized (100,000 longitude/latitude of world country boundaries)	$30
KST Fonts (13,200 characters in 139 mixed fonts: specify TEX or bitmap format)	$30
USNO Floppy Almanac (high-precision moon, sun, planet & star positions)	$20
NBS Hershey Fonts (1,377 stroke characters in 14 fonts)	$15
U. S. Map (15,701 points of state boundaries)	$15

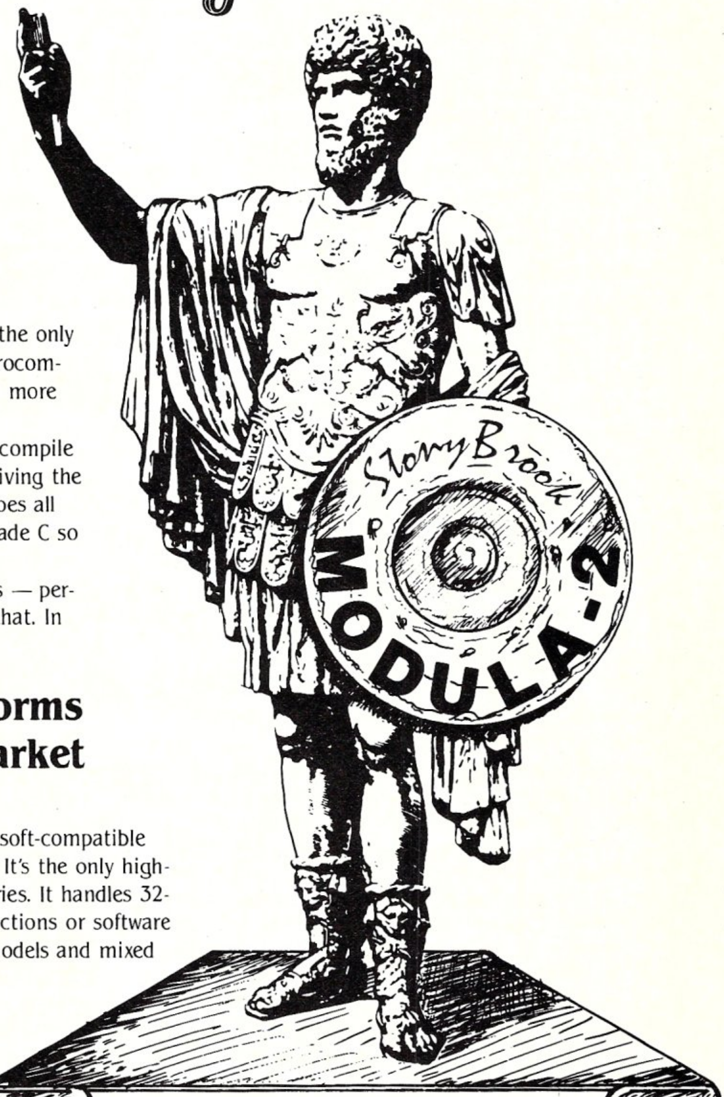
The Austin Code Works
11100 Leafwood Lane
Austin, Texas 78750-3409 USA

acw!info@uunet.uu.net

Voice: (512) 258-0785
BBS: (512) 258-8831
FidoNet: 1:382/12

Free shipping on prepaid orders **For delivery in Texas add 7%** **MasterCard/VISA**

I have come to bury C, sir, not to praise it.



C served us well in the days of kilobytes and kilohertz. It was the only language we could implement efficiently on our newborn microcomputers. But with today's mega-machines, shouldn't we demand more from our compilers?

Modula-2 increases productivity by catching your errors at compile time. You'll easily modularize and structure your programs, driving the hordes of barbaric bugs into the hinterlands. And Modula-2 does all this without taking away the low-level machine access that made C so popular.

Until now, you had to pay a price for the Modula-2 advantages — performance just didn't measure up to C. But we've changed all that. In a suite of benchmarks developed by PC Week:

Stony Brook Modula-2 outperforms the best C compilers on the market

(and no other Modula-2 compiler even comes close).

Stony Brook Modula-2, for 80x86 machines, produces Microsoft-compatible objects, and fully supports both Microsoft Windows and OS/2. It's the only high-level language compiler that lets you write dynamic link libraries. It handles 32- and 64-bit real numbers with in-line 80x87 coprocessor instructions or software emulation. And Stony Brook Modula-2 supports six memory models and mixed model programming.

You might want to bury your C compiler once you have used Stony Brook Modula-2, but you won't have to. We made it possible to directly call C and other languages from Modula-2, so you won't have to throw away your investment in C code.

So, friends, programmers, and C-users, lend us your ears. Call us or write for more information and to find out how you can get a demo compiler.

Benchmark	Stony Brook M-2 V1	Logitech M-2 V3	Turbo C V1	Microsoft C V4
Magnifier	0.05	0.06	0.05	0.05
For Loop	0.28	0.73	0.44	0.68
While Loop	0.35	0.75	0.49	0.66
Repeat Loop	0.33	0.68	0.38	0.66
Literal Assign	0.33	0.97	0.60	0.94
Memory Access	0.33	1.04	0.55	1.00
Real Arithmetic	4.68	13.40	3.73	2.96
Real Algebra	4.18	12.87	3.07	3.64
Vector	0.77	1.75	1.21	1.26
Equal If	0.66	1.47	0.93	1.32
Unequal If	0.66	1.43	0.99	1.32
No Parameters	0.33	0.60	0.34	1.04
Value	0.50	0.71	0.60	1.21
Reference	0.61	0.77	0.66	1.21
Real Maths	3.90	4.61	7.85	18.51

The compiler package includes DOS runtime library objects and full sources for our split-screen text editor for $195. The development package includes all of the above plus an automatic make utility, a symbolic debugger, and run-time library sources for $345. MasterCard and Visa accepted. Add $5 for shipping in North America, $25 for overseas shipping.

StonyBrook INC. SOFTWARE

Forest Road, Wilton, New Hampshire 03086 ● (603)654-2525 Ask for Cleopatra.

How to tell the difference between DESQview™ 2.0 and any other environment.

Selecting DESQview, the environment of choice, can give you the productivity and power you crave, without the loss of your old programs and hardware. If you like your existing programs, want to use them together, transfer data between them, print, sort, communicate with or process-in-background, yet still have the need to keep in place your favorite PC(8088, 8086, 80286 or 80386), DESQview is the "proven true" multitasking, multiwindowing environment for you. Best of all, DESQview 2.0 is here now, with all the money saving, time saving, and productivity features that others can only promise for the all-too-distant future.

One picture is worth a thousand promises.

And with DESQview's new graphics enhancements for Hercules, CGA, EGA, and VGA, Version 2.0 still offers the same award winning and pioneering features for programs that earned DESQview its leadership, only now you can also run desktop publishing programs, CAD programs, even GEM-™, Topview-™, and Microsoft Windows-™ specific programs. In some cases you'll add as little as 10–40K to your system overhead. Now you can have multi-tasking, multi-windowing, break the 640K habit too and still get an auto dialer, macros, menus for DOS and, for advanced users, a new complete application programmer's interface capability. No wonder that over the years, and especially in recent months, DESQview, and now DESQview 2.0 have earned extravagant praise from some of the most respected magazines in the industry.

INFO WORLD
REPORT CARD
DESQview 2.0
9.1
July 6, 1987

"Product of the Year" by readers vote in InfoWorld.

"Best PC Environment" by popular vote at Comdex Fall in PC Tech Journal's "System Builder" Contest.

"—I wouldn't want to run an IBM or compatible computer without DESQview"—InfoWorld, Michael Miller.

"A colossus among windowing environments"… "will run almost anything"—PC Week, Marvin Bryan.

"Windows, promises, but DESQview delivers"—MICROTIMES, Birell Walsh.

No other environment has consistently pioneered features, openness, and productivity. See for yourself. Send in the coupon. The possibilities are endless with DESQview 2.0.

Attention Programmers: For more information about Quarterdeck's API, and future 386 program extensions, call us today.

SYSTEM REQUIREMENTS
IBM Personal Computer and 100% compatibles (with 8086, 8088, 80286 or 80386 processors) with monochrome or color display; IBM Personal System/2 • Memory: 640K recommended; for DESQview itself 0-145K • Expanded Memory (Optional): expanded memory boards compatible with the Intel AboveBoard; enhanced expanded memory boards compatible with the AST RAMpage • Disk: Two diskette drives or one diskette drive and a hard disk • Graphics Card (Optional): Hercules, IBM Color/Graphics (CGA), IBM Enhanced Graphics (EGA), IBM Personal System/2 Advanced Graphics (VGA) • Mouse (Optional): Mouse Systems, Microsoft and compatibles • Modem for Auto-Dialer (Optional): Hayes or Compatible • Operating System: PC-DOS 2.0-3.3; MS-DOS2.0-3.2 • Software: Most PC-DOS and MS-DOS application programs; programs specific to TopView 1.1, GEM 1.1 and Microsoft Windows 1.03 • Media: DESQview 2.0 is available on either 5¼" or 3½" floppy diskettes

Rush me DESQview 2.0! Today!

No. of Copies	Media 3½"/5¼"	Product	Retail Price ea.	Total
		DESQview 2.0	$129.95	$
	Shipping & Handling	USA Outside USA	$ 5.00 $ 10.00	$
		Sales Tax (CA residents)	6.5%	$
Payment: ☐ Visa ☐ MC ☐ AMEX ☐ Check			Amount Enclosed	$

Credit Card: Valid Since ______ / ______ Expiration ______ / ______

Card Number: ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Credit Card Name ______

Shipping Address ______

City ______ State ______ Zip ______ Telephone ______

Mail to: Quarterdeck Office Systems, 150 Pico Boulevard, Santa Monica, CA 90405.
NOTE: If you own DESQview call us for a special upgrade offer, or send in your DESQview registration card. AST Special Edition users included.

DESQview™
Quarterdeck

Quarterdeck Office Systems • 150 Pico Boulevard, Santa Monica, CA 90405 • (213) 392-9851

How A C Programmer Became A Screen Star

Screens, the Visible Part of Your Program.

A program is often judged by how well the screens are executed. However, the real creativity lies in what goes on behind the screens.

ScreenStar is a product that allows your real creativity to light up the screen. It reduces costly screen, window, and data validation development time.

You Take the Bows, We Write the Code.

Our natural drawing commands allow you to paint any screen imaginable. Press one key when you are satisfied and ScreenStar produces concise, commented, ready-to-compile code. This allows immediate testing of the I/O screens, including smooth, even scrolling between multiple screens.

Create or capture complex screens with data-entry filters built in.

If all ScreenStar did was turn screens into code it would be a useful tool. Yet ScreenStar also permits a wide range of field types. Some of the choices include date, alphanumeric, telephone, yes/no, dollar, time and user-definable fields.

Other valuable data-entry filters are built in, such as required field, display only, and many others. All screen fields are generated with error-checking routines.

ScreenStar Not Only Captures Your Imagination, It Captures Screens.

The memory-resident capture program converts any screen into a ScreenStar file in seconds, including those generated by programs like Dan Bricklin's Demo Program.

ScreenStar Sets the Stage for Windows.

ScreenStar comes with a complete window generating library. You design the help screens and pop-up windows. Essential ScreenStar windowing functions tie them together in one smooth package.

Curtain Call.

They may not ask for your autograph, but they will want to know how you did those screens. Screenstar is more than a screen-painting program. It is a screen processor. No professional programming environment will be complete without this product.

We know you will enjoy using ScreenStar. However, should you give it less than rave reviews, return it within 30 days for a full refund.

★ Interactive screen painting and subsequent code generation.

★ Multiple screen design and scrolling.

★ TSR screen capture program, works with any program including Dan Bricklin's Demo Program.

★ Complete window design including overlapping window functions.

★ Screens are compressed into data structures, and remain a permanent part of the program. No messy data files to look for.

Price - $99

W/Source add $99

Audition Our Product Today. Call: